digital

POLYCENTER Software Installation Utility Developer's Guide

Part Number: AA-Q28MA-TK

POLYCENTER Software Installation Utility Developer's Guide

Order Number: AA–Q28MA–TK

March 1994

This manual describes how to package software products using the POLYCENTER Software Installation utility. It describes the product description language, product description files, product text files, and other relevant concepts.

Revision/Update Information: This is a new manual.

Software Version: OpenVMS AXP Version 6.1
OpenVMS VAX Version 6.1

Digital Equipment Corporation
Maynard, Massachusetts

March 1994

The postpaid Reader's Comments forms at the end of this document request your critical evaluation to assist in preparing future documentation.

ZK5952

This document is available on CD–ROM.

This document was prepared using VAX DOCUMENT Version 2.1.

Send Us Your Comments

We welcome your comments on this or any other OpenVMS manual. If you have suggestions for improving a particular section or find any errors, please indicate the title, order number, chapter, section, and page number (if available). We also welcome more general comments. Your input is valuable in improving future releases of our documentation.

You can send comments to us in the following ways:

- Internet electronic mail: OPENVMSDOC@ZKO.MTS.DEC.COM
- Fax: 603-881-0120 Attn: OpenVMS Documentation, ZKO3-4/U08
- A completed Reader's Comments form (postage paid, if mailed in the United States), or a letter, via the postal service. Two Reader's Comments forms are located at the back of each printed OpenVMS manual. Please send letters and forms to:

  Digital Equipment Corporation
  Information Design and Consulting
  OpenVMS Documentation
  110 Spit Brook Road, ZKO3-4/U08
  Nashua, NH 03062-2698
  USA

You may also use an online questionnaire to give us feedback. Print or edit the online file SYS$HELP:OPENVMSDOC_SURVEY.TXT. Send the completed online file by electronic mail to our Internet address, or send the completed hardcopy survey by fax or through the postal service.

Thank you.

Contents

Part II Product Description Language Statements

A Migrating from VMSINSTAL to the POLYCENTER Software Installation Utility

B Advanced PDF Concepts

Glossary

Index

Examples

Figures

Tables

Preface

Intended Audience

This manual is intended for individuals who are responsible for packaging software products.

Document Structure

This manual is organized into two parts, two appendixes, and a glossary as follows:

- Part I contains the chapters that describe the concepts you should be familiar with to create software kits with the POLYCENTER Software Installation utility. It also contains instructions on how to write a product description file and product text file. Part I contains the following chapters:
  - Chapter 1 describes utility concepts.
  - Chapter 2 describes writing the product description file. It also contains sample product descriptions.
  - Chapter 3 describes writing the product text file. It also contains sample product text files.
  - Chapter 4 describes how to package your product.
- Part II describes product description language statements.
- Appendix A contains information about migrating from the VMSINSTAL utility to the POLYCENTER Software Installation utility.
- Appendix B describes some advanced concepts such as managed object scope and lifetime.
- The Glossary lists and defines POLYCENTER Software Installation utility terminology.

Associated Documents

The *POLYCENTER Software Installation Utility User's Guide* describes the tasks that system managers perform using the POLYCENTER Software Installation utility. It explains operations such as software installation and removal. It also demonstrates the two POLYCENTER Software Installation user interfaces: the DIGITAL Command Language (DCL) and DECwindows Motif.

Conventions

In this manual, every use of OpenVMS AXP means the OpenVMS AXP operating system, every use of OpenVMS VAX means the OpenVMS VAX operating system, and every use of OpenVMS means both the OpenVMS AXP operating system and the OpenVMS VAX operating system.

In this manual, every use of DECwindows and DECwindows Motif refers to DECwindows for Motif software.

This manual also uses the following conventions:

. . .	Horizontal ellipsis points in examples indicate one of the following possibilities: • Additional optional arguments in a statement have been omitted. • The preceding item or items can be repeated one or more times. • Additional parameters, values, or other information can be entered.
( )	In command format descriptions, parentheses indicate that, if you choose more than one option, you must enclose the choices in parentheses.
[ ]	In command format descriptions, brackets indicate optional elements. You can choose one, none, or all of the options. (Brackets are not optional, however, in the syntax of a directory name in an OpenVMS file specification or in the syntax of a substring specification in an assignment statement.)
{ }	In command format descriptions, braces surround a required choice of options; you must choose one of the options listed.
boldface text	Boldface text represents the introduction of a new term or the name of an argument, an attribute, or a reason (user action that triggers a callback). Boldface text is also used to show user input in Bookreader versions of the manual.
italic text	Italic text indicates POLYCENTER Software Installation utility statements, emphasizes important information, indicates complete titles of manuals, and indicates variables. Variables include information that varies in system messages (Internal error *number*), in command lines (/PRODUCER=*name*), and in command parameters in text (where *device-name* contains up to five characters).
UPPERCASE TEXT	Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
-	A hyphen in code examples indicates that additional arguments to the request are provided on the line that follows.
numbers	All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.
parameters	Unless otherwise noted, all parameters that you use with product description language (PDL) statements are strings either enclosed in quotation marks or not enclosed in quotation marks.

Part I

Concepts

Part I describes the concepts you should be familiar with to create software kits. It also contains instructions for writing product description and text files.

1 Overview

The POLYCENTER Software Installation architecture is a new technology that simplifies the distribution and management of software.

For system managers, the POLYCENTER Software Installation utility provides DIGITAL Command Language (DCL) and DECwindows Motif interfaces that they can use to configure, install, and remove software products. It also allows system managers to track the status of software on their systems.

For software providers, the POLYCENTER Software Installation utility simplifies the task of packaging software because:

- The utility provides a standard process and toolset for packaging products on all platforms.
- All software and documentation can exist on a single piece of media.
- The utility keeps track of which products (and versions) have been installed and removed in the execution environment. Using this information, you can design your installation procedure so that system managers can use the utility to manage version dependencies.
- Installations require less code than most conventional installation methods. This results in performance gains and reduced development time over conventional installations. The code is also nonprocedural, which means that you do not need to understand the details of how the utility functions.
- You can include installation text to guide users through an installation, resulting in a higher installation success rate.

1.1 Packaging a Product

To package your product using the POLYCENTER Software Installation utility, you usually create the following three components:

- **Product description file (PDF)**. The PDF is a text file that specifies the execution environment for your product. For example, if your product provides certain files and directories, you must specify this in the product description file. Minimum versions of hardware and software are examples of other aspects of the execution environment you can specify in the product description file.

  You specify these aspects of the execution environment in the PDF using the **product description language (PDL)**. The PDL consists of the statements described in Part II.
- **Product text file (PTF)**. The PTF contains all the product-specific text that the utility can display during product manipulation (for example, text displayed to users during an installation). Certain statements in the PDF require text in the PTF.

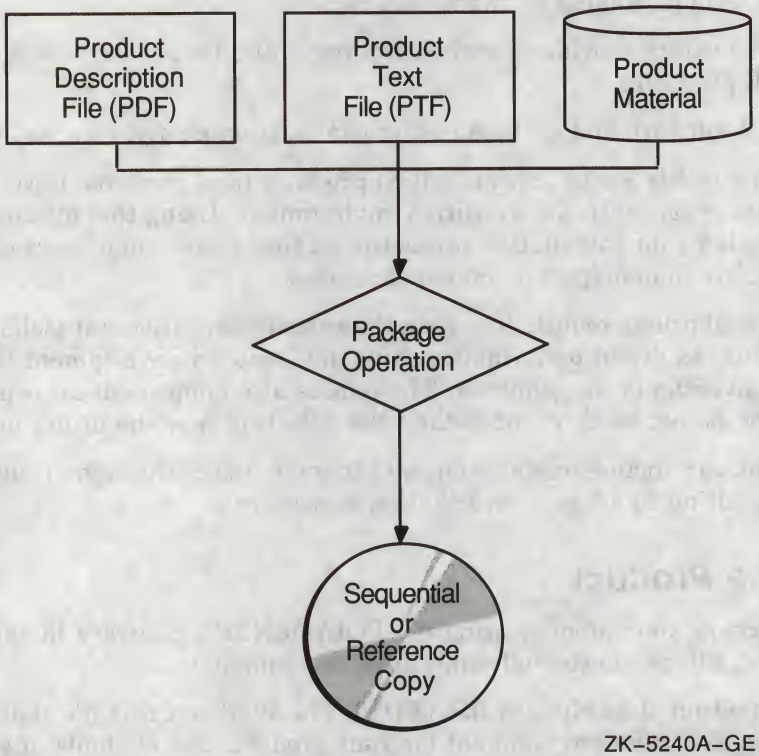
- **Product material**. The product material consists of the files associated with the product.

After you create these components, you can perform a **package operation**. The package operation creates a ready-to-install version of your product (that the utility can access) in one of the following formats:

- **Reference copy**. This is the standard form your product takes as a result of a copy or package operation. It is in a directory tree on a random-access device (for example, a compact disc drive).
- **Sequential copy**. This is an optional form your product takes as a result of a copy or package operation. It consists of a container file that can be placed on a sequential-access device (such as a magnetic tape drive).

Figure 1–1 shows how the package operation uses the PDF, PTF, and product material to create a reference or sequential copy.

Figure 1–1 Package Operation



Although you provide the PDF, PTF, and product material as input to the package operation, they also exist in the reference or sequential copy that you create. The package operation changes the format of the output PTF (for more information, see Section 3.2). The output PDF is in the same format as the input PDF, but the utility may modify statements in the output PDF. For example, the package operation adds the size option to *file* statements in the output PDF.

1.2 Packaging a Platform

In addition to packaging individual products, the POLYCENTER Software Installation utility gives you the means to assemble **integrated platforms**. An integrated platform is a combination of several products.

Figure 1–2 shows an example of an integrated platform.

Figure 1–2 Integrated Platform Example

ZK–5242A–GE

To package a platform, you create a **platform PDF** and **platform PTF**. In addition to other statements, the platform PDF can contain *software* statements that specify the products that make up the platform. The individual products have their own PDFs and PTFs (independent of the platform PDF and PTF). For more information about platform PDFs, see Section 2.5.

1.3 System Management

System managers use the POLYCENTER Software Installation utility to install software products, track their environment, remove software, and make product configuration changes.

The utility records events such as product installation and removal in a repository called the **product database (PDB)**. You can query the product database by entering the DCL command PRODUCT SHOW or by choosing Show History or Show Object from the DECwindows Motif Mode menu. You can obtain information about which products and versions are installed on the system and the history of product activity on the system as well as obtain information about specific files on the system.

Different parts of the utility interact with each other and with the underlying operating system. For example, when you install a product using the POLYCENTER Software Installation utility, the following occurs:

- The utility places all the files and objects for the product as specified in the PDF (using the underlying operating system's file and management services in the process). The utility also interacts with the PTF, displaying text during an installation.

- The product database records the installation. Users can query this database to determine the status of products. If the need arises to remove a product, the utility removes all the files and objects associated with the product and removes the product from the database. The product database contains the history of the product's activity from installation to removal. You can use the product database to determine the status of a product, the history of a product, and what configuration choices were made.

1.4 Managed Objects

Managed objects exist to support the proper functioning of your product. Files, directories, and accounts are all types of managed objects. An executable image is an example of a file managed object.

Using PDL statements, you can specify the names and properties of the managed objects that are necessary for your product. At installation time, the utility uses your product description file (PDF) to create the managed objects for your product. At removal, the utility uses the PDF to remove the managed objects from the environment.

Most PDL statements create corresponding managed objects. For example, you use the *file* statement to specify a file managed object. In contrast, **utility directives** are PDL statements that do not specify managed objects. Utility directives affect the operation of the utility but do not affect the execution environment. For example, the *error* statement is a utility directive; it displays a text module to the user and does not affect the execution environment.

1.4.1 Managed Object Conflict

Occasionally, your product will supply a managed object that conflicts with another managed object (for example, two products supplying files with identical names). When the utility detects conflict, it displays an informational message.

The following statements detect managed object conflict and display informational messages:

- *account*
- *bootstrap block*
- *directory*
- *file*
- *link*
- *loadable image*
- *module*
- *network object*
- *patch image*
- *patch text*
- *register module*
- *rights identifier*
- *software*

In some cases, the POLYCENTER Software Installation utility allows you to anticipate and resolve conflict before it occurs. For example, the generation option to the *file* statement lets you resolve managed object conflict. During installation, if the utility attempts to create a file that already exists, it compares the generation numbers of the files (if present), preserving the file with the highest generation number. The following statements provide some level of conflict resolution:

- *directory*
- *file*
- *module*
- *register module*

The description of these statements in Part II indicates how each one resolves managed object conflict.

1.4.2 Managed Object Replacement and Merging

As described in Section 1.4.1, managed objects occasionally have characteristics that conflict with each other. However, your product can supply managed objects that have different characteristics but do not conflict. The POLYCENTER Software Installation utility handles this situation differently depending on the kit type.

For full, operating system, and platform kit types, the utility deletes the existing object and replaces it with the object and characteristics provided by the new version of the product. For partial, patch, and mandatory update kit types, the utility preserves the characteristics of existing objects. For example, the security environment you establish for your product is preserved when you install a partial, patch, or mandatory update kit.

If you want to provide new characteristics for a managed object in a partial, patch, or mandatory update kit, use the *remove* statement to delete the existing object and then respecify it with the desired characteristics.

For more information about kit types, see Table 2–1.

1.4.3 Managed Object Scope and Lifetime

The **scope** of a managed object defines the degree of sharing that the managed object permits. For example, some objects are available only to certain processes, and some can be shared by all processes. The utility usually ensures that managed objects have the correct scope.

Occasionally, an advanced PDF writer might need to use the *scope* statement to give a managed object a scope other than its default. For more information about specifying the scope of a managed object, see Appendix B.

2

Creating the Product Description File

Creating the product description file (PDF) is a two-step process:

1. Determine the required characteristics of the execution environment for your product or platform. For example, you must determine your product's files and required operating characteristics.
2. Translate those characteristics into the product description language (PDL) statements in the PDF.

This chapter describes this process. Chapter 3 describes how to create a corresponding product text file (PTF).

2.1 General Guidelines

The POLYCENTER Software Installation utility is optimized for software that executes directly from the reference copy. It is also intended to simplify the job of system managers, making products quick and easy to install and manage. For these reasons, use the following guidelines when writing PDFs:

- Minimize installation activity (such as linking images and building databases). Instead, include all material required for product execution on the reference copy.
- Make your products adapt to the target environment at execution time rather than installation time. This practice keeps products consistent across varying configurations.
- Avoid requiring system parameter settings on the target system that would require rebooting the system.
- Minimize configuration choices at installation time.
- Ensure that the PDF expressess all the known requirements that your product needs to execute. Use the checklist in Section 2.2 to define the requirements for the target environment.

2.2 Defining Your Environment

To define the environment for your product, use the following checklist. (Part II of this manual describes each PDL statement.)

☐ **Does your product depend on other software?**

For example, your product may require a specific version of the operating system or optional software products. To express these software requirements, use the *software* statement or function. Note that software you reference with a *software* statement must be registered in the product database to be recognized by the POLYCENTER Software Installation utility. If you install a product using a mechanism other than the POLYCENTER Software Installation utility, the product database does not store information about

the product unless you register a full or **transition product description**. For more information about creating transition product descriptions, see Section 2.6. For more information about registering a product, see the *POLYCENTER Software Installation Utility User's Guide*.

☐ **If you are creating a platform, what software products comprise the platform?**

If you are creating a platform, you must specify the software products that make up the platform. To specify the products that comprise your platform, use the *software* statement with the component option.

☐ **Does your product require specific hardware devices?**

For example, your product may require that the system have access to certain peripheral devices, such as a compact disc drive or printer. To display a message to users expressing these hardware requirements, use the *hardware device* statement.

☐ **Does your product run only on specific computer models?**

Some products run only on certain computer models. For example, recent versions of the OpenVMS operating system are no longer supported on the VAX 11–725 computer. If this is the case with your product, use the *hardware processor* statement to display a message to users.

☐ **Does your product require specific images, files, or directories?**

All the files, images, and directories that your product requires should be expressed in *file* or *directory* statements.

☐ **Does your product require a special account on the system?**

Some products require a dedicated account on the system. If this is the case for your product, use the *account* statement to supply the account.

☐ **Does your product require network objects?**

Some products require network objects on the system. If this is the case for your product, use the *network object* statement to supply the required network objects.

☐ **Do you want to set up rights identifiers?**

If you want to set up rights identifiers for your product, use the *rights identifier* statement.

☐ **Does your product supply an image to the system loadable images table?**

To supply an image to the system loadable images table, use the *loadable image* statement.

☐ **Does your product have several options that the user can choose?**

Although it is a good practice to limit the number of options that the user can choose, you may need to present the user with options during installation. To present options to the user, use the *option* statement.

☐ **Do you need to patch an executable image?**

To patch an executable image, use the *patch image* statement.

☐ **Do you need to patch a text file?**

To patch a text file, use the *patch text* statement.

☐ **Does your product have specific security requirements?**

If the files and directories for your product require special protection or access controls, you can express this in the product description. See the descriptions of the *directory* statement and the *file* statement. You can also supply a rights identifier using the *rights identifier* statement.

☐ **Does your product require certain values for system parameters?**

Many software products require that system parameters have certain values for the product to function properly. Use the *system parameter* statement to display system parameter requirements to users.

☐ **Does your product require certain values for process parameters?**

If your product requires certain values for process parameters, use the *process parameter* statement to display these requirements to users.

☐ **Does your product require certain values for process privileges?**

If your product requires certain process privileges, use the *process privilege* statement to display these requirements to users.

☐ **Do you want to include a functional test with your product?**

If you have a functional test for your product, you can include it in the product material to verify that your product installed correctly. To execute the functional test for your product, use the *execute test* statement.

☐ **Are there commands that your installation procedure needs to execute that are outside the domain of the POLYCENTER Software Installation utility?**

If you have commands that your installation procedure needs to execute that have no POLYCENTER Software Installation utility equivalent, use the *execute* statement.

☐ **Does your product have specific pre- or postinstallation tasks?**

You can use the POLYCENTER Software Installation utility to automate these tasks; however, there may be some tasks you want users to perform that are outside the capabilities of the utility. You can inform users of such tasks using the *information* statement. You can also use several of the *execute* statements to perform these tasks.

☐ **Does your product require command, help, macro, object, or library modules?**

You should express the following types of modules in your PDF:

- DIGITAL Command Language (DCL) command definition modules
- DCL help modules
- Macro modules
- Object modules
- Text modules

You can express these types of modules using the *module* statement.

☐ **What happens to existing product files?**

You should make sure that your product's files are handled correctly during an installation or upgrade. The POLYCENTER Software Installation utility deletes obsolete files that are replaced when you install a full, operating system, or plaform kit. In partial, patch, and mandatory update kits, the existing files are preserved. To remove obsolete files, use the *remove* statement and *file* statement options.

☐ **Does your product require documentation?**

You may want to include online documentation (such as release notes) with your product. To express the documentation requirements for your product, use the release notes option to the *file* statement.

2.3 PDF File Name Format

After you have determined the requirements for your product, use a text editor to create the PDF. You can use any valid OpenVMS file name and file type for the PDF. You supply the PDF as input to the package operation. The PDF remains in your directory. When you package the kit, the utility creates the kit, including an output PDF with a file name that has the following format:

producer-ABI-product-version-kit_type.PCSI$DESCRIPTION

where:

- *producer* is the legal owner of the software product. For example, for Digital software products, this part of the PDF file name is DEC.

- *ABI* is the **Application Binary Interface (ABI)**, which specifies the hardware and software combination that the product requires. For example, OpenVMS layered products that execute on OpenVMS VAX hardware use VAXVMS in this part of the PDF file name. OpenVMS layered products that execute on OpenVMS AXP hardware use AXPVMS in this part of the PDF file name.
- *product* is the name of the software product. The combination of *producer*, *ABI*, and *product* must be unique. Note that you cannot specify hyphens (-) in the product name.
- *version* is the version identifier of the product in the following format:

  tvvuu-ep

  where:

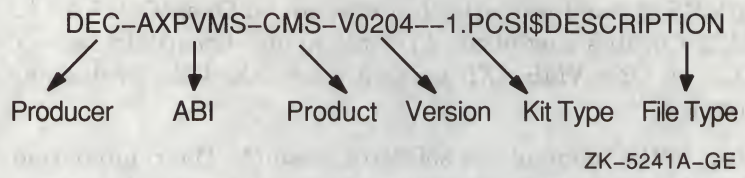
  - *t* is a single uppercase letter (for example, V) that indicates the version type.
  - *vv* is a decimal integer from 00 through 99 that identifies a major software release.
  - *uu* is a decimal integer from 00 through 99 that identifies a minor release.
  - *e* is an optional positive decimal integer you can use to indicate the edit level of software. Separate the edit level from the rest of the version identifier with a hyphen (-). The hyphen is required *even if you do not specify an edit level.*
  - *p* is any string other than a positive decimal integer that indicates a patch level (for example "MUP"). This string is optional.
- *kit_type* is a decimal integer from 1 through 7 that specifies a PDF type listed in Table 2–1.

Table 2–1 PDF Kit Types

Kit Type	Meaning
1	Indicates a **full** PDF, which describes application software that is not operating system software.
2	Indicates an **operating system** PDF, which describes operating system software.
3	Indicates a **partial** PDF, which describes a software version update (software upgrade) to existing software in the target environment.
4	Indicates a **patch** PDF, which describes a software update to existing software in the target environment but does not change the version level.
5	Indicates a **platform** PDF, which describes an integrated platform consisting of several products.
6	Indicates a **transition** PDF, which describes a product that has not been converted to the POLYCENTER Software Installation utility.
7	Indicates a **mandatory update** PDF, which describes a software update that you must apply to your system before a version update is complete.

Figure 2–1 shows an example of an output PDF file name.

Figure 2–1 Output PDF File Name Example



Note

Because this example does not have an edit level in the version identifier, there are two hyphens between the version identifier and the kit type.

2.4 Writing PDL Statements

Begin the PDF with a *product* statement that provides information about your product. For the remainder of the PDF, use the PDL statements in Part II of this manual to describe the execution environment for your product. End the PDF with the *end product* statement.

2.4.1 PDF Format

When writing the PDF, use two hyphens as a comment character. When the utility encounters the comment character, it ignores the rest of the text on the line.

The following PDL statements are group statements:

- *if*
- *option*
- *part*
- *product*
- *remove*
- *scope*

Group statements have a required terminator (for example, *end option*) and enclose several other statements. For example, *file* statements enclosed within a *remove* statement and an *end remove* statement are in a remove group.

All PDL statements must be terminated by a semicolon (;). For example:

```
option example ;
  file [SYSHLP.EXAMPLES]product_example.txt ;
  file [SYSHLP.EXAMPLES]product_example_b.txt ;
end option ;
```

2.4.2 PDL Base Data Types and Values

The PDL has several base data types that you must use when passing parameters to the PDL statements listed in Part II. Table 2–2 describes the PDL base data types and their values.

Table 2–2 Base Data Types and Values

Data Type	Values
Boolean	The number 0 (false), the number 1 (true), the keywords **false**, **true**, **yes**, and **no**.
String	ISO Latin-1 characters with a minimum length of 0 characters. The maximum length of a string is defined by the file system where the POLYCENTER Software Installation utility executes. Table 2–3 lists the additional contraints on PDL strings.
Signed integer	Specifies a positive, negative, or zero integral value.
Unsigned integer	Specifies a zero or positive integral value.
Version identifier	See the description in Section 2.3.
Text module name	Specifies a unique name for a text module using the printable ISO Latin-1 characters, excluding horizontal tab, space, exclamation point, and comma.

Table 2–3 describes additional contraints on the string data type.

Table 2–3 String Data Type Constraints

String Type	Values	Examples
Unconstrained	None; any character may appear in any position.	
Access control entry (ACE)	Specifies an ACE for a directory or file.	(IDENTIFIER=[DOC,RUGGLES],ACCESS=READ)
Command	Specifies an operating system command that you want to execute during a specific operation.	@sys$test:prod$ivp.com
Device name	Specifies the name of a hardware device.	DUB6:
File name	Specifies a file name (without a device or directory specification).	STARTUP.DAT
Identifier name	Specifies a rights identifier.	DOC
Module name	Specifies the name of a module in a library.	FMSHELP

(continued on next page)

Table 2–3 (Cont.) String Data Type Constraints

String Type	Values	Examples
Processor model name	Specifies the model identification of a particular computer system.	7
Relative directory specification	Specifies the directory name and, if necessary, the directory path, relative to the root directory path.	[SYSUPD.MY_PRODUCT]
Relative file specification	Specifies the directory path and file name, relative to the root directory path.	[SYSUPD.MY_PRODUCT]DRIVER.DAT
Root directory specification	Specifies the directory name and a trailing period (.). If you specify a directory name and omit the period, it is inserted. If necessary, you can add the device name.	[SYSUPD.]

2.4.3 PDF Examples

Example 2–1 shows a PDF for Checktran. You can see how the requirements of the execution environment were translated into statements in the PDF.

Example 2–1 Checktran Product Description File

```
$ TYPE CHECKTRAN-TEST.PDF

product DEC VAXVMS CHECKTRAN V4.3 full ; ❶
    software DEC VAXVMS VMS version minimum V5.0 ; ❷
    file [SYSEXE]CHECKTRAN.EXE ; ❸
    module [SYSUPD]CHECKTRAN.CLD type command module CHECK ; ❹
    module [SYSUPD]CHECKTRAN.HLP type help module HELP ; ❺
    file [SYSTEST]CHECKTRAN$IVP.COM ; ❻
    file [SYSEXE]CHECKTRAN.DAT ;
    execute test @SYS$TEST:CHECKTRAN$IVP ; ❼
end product ; ❽
```

The following list describes the function of the PDL statements in Example 2–1:

❶ The *product* statement identifies the product as DEC VMS CHECKTRAN Version 4.3. The full option specifies that the kit is a complete software distribution and not an update or patch.

❷ The *software* statement specifies that to run Checktran, the execution environment must be running at least VMS Version 5.0.

❸ The first *file* statement in the PDF supplies the file [SYSEXE]CHECKTRAN.EXE.

❹ The first *module* statement in the PDF installs the command module for Checktran in the default command library [SYSLIB]DCLTABLES.EXE.

❺ The second *module* statement in the PDF installs the help module for Checktran in the default help library [SYSHLP]HELPLIB.HLB.

❻ The next statement installs the DCL command procedure that will be used for the functional test.

❼ The *execute test* statement executes the functional test for the product.

❽ The *end product* statement closes the group of product statements.

In this example, the PDF for Checktran is relatively brief. Example 2–2 shows a PDF for DEC TCP/IP Services for OpenVMS, which is more complex.

Example 2–2 UCX Product Description File

```
$ TYPE UCX.DES

-- "DEC TCP/IP Services for VMS" ❶

product DEC VAXVMS UCX V2.0 full ; ❷

software DEC VAXVMS VMS version minimum V5.4 ; ❸

process parameter ASTLM minimum 24 ; ❹
process parameter BIOLM minimum 18 ;
process parameter BYTLM minimum 32768 ;
process parameter DIOLM minimum 18 ;
process parameter ENQLM minimum 200 ;
process parameter FILLM minimum 100 ;

rights identifier UCX$NFS_REMOTE ; ❺

execute start "@SYS$STARTUP:UCX$STARTUP.COM" ❻
        stop "@SYS$STARTUP:UCX$SHUTDOWN.COM" ;

execute test "@SYS$TEST:UCX$IVP.COM" ; ❼

information PRE_INSTALL confirm ; ❽

information POST_INSTALL phase after ; ❾

directory [SYSTEST.UCX] ; ❿
directory [SYSHLP.EXAMPLES.UCX] ;

module [000000]UCX$TOP_LEVEL_HELP.HLP type help module UCX ; ⓫
module [000000]UCX.CLD type command module UCX ; ⓬

file [SYSEXE]UCX$SERVICE.DAT ; ⓭
file [SYSEXE]UCX$FTPSERVER.COM ;
file [SYSEXE]UCX$SNMP_AGENT.EXE ;
file [SYSEXE]UCX$UCP.EXE ;
   .
   .
   .
file [SYSTEST]UCX$IVP.COM ;
file [SYSTEST.UCX]UCX$INET_IVP.EXE ;
file [SYSUPD]UCX$CLEANUP.COM ;

option EXAMPLES ; ⓮
```

(continued on next page)

Example 2–2 (Cont.) UCX Product Description File

```
  file [SYSHLP.EXAMPLES.UCX]UCX$IOCTL_ROUTINE.C ;
  file [SYSHLP.EXAMPLES.UCX]UCX$TCP_CLIENT_IPC.C ;
  file [SYSHLP.EXAMPLES.UCX]UCX$TCP_CLIENT_QIO.C ;
  file [SYSHLP.EXAMPLES.UCX]UCX$TCP_CLIENT_QIO.MAR ;
  file [SYSHLP.EXAMPLES.UCX]UCX$TCP_SERVER_IPC.C ;
  file [SYSHLP.EXAMPLES.UCX]UCX$TCP_SERVER_QIO.C ;
  file [SYSHLP.EXAMPLES.UCX]UCX$TCP_SERVER_QIO.MAR ;
  file [SYSHLP.EXAMPLES.UCX]UCX$UDP_CLIENT_IPC.C ;
  file [SYSHLP.EXAMPLES.UCX]UCX$UDP_CLIENT_QIO.C ;
  file [SYSHLP.EXAMPLES.UCX]UCX$UDP_CLIENT_QIO.MAR ;
  file [SYSHLP.EXAMPLES.UCX]UCX$UDP_SERVER_IPC.C ;
  file [SYSHLP.EXAMPLES.UCX]UCX$UDP_SERVER_QIO.C ;
  file [SYSHLP.EXAMPLES.UCX]UCX_SECURITY_DRIVER.MAR ;
  file [SYSHLP.EXAMPLES.UCX]BUILD_UCX_SECURITY_DRIVER.COM ;
  file [SYSHLP.EXAMPLES.UCX]BGDRIVER_SEC.EXE ;
  file [SYSHLP.EXAMPLES.UCX]UCX_TRACE.EXE ;
  file [SYSHLP.EXAMPLES.UCX]TRACEROUTE.EXE ;

end option ;

option NFS ; (15)

  file [SYSEXE]UCX$CONVERT.FDL ;
  file [SYSEXE]UCX$CONVERT.COM ;
  file [SYSEXE]UCX$SERVER_NFS.EXE ;

  file [SYSHLP]UCX$VMS_FILES.DOC ;

  file [SYSLIB]UCX$CFS_SHR.EXE ;

end option ;

option APPLICATIONS ; (16)

  module [000000]FTP.CLD type command module FTP ;
  module [000000]RLOGIN.CLD type command module RLOGIN ;
  module [000000]RSH.CLD type command module (RSH,RSHELL) ;
  module [000000]TELNET.CLD type command module TELNET ;
  module [000000]UCX$LPQ_CLD.CLD type command module LPQ ;
  module [000000]UCX$LPRM_CLD.CLD type command module LPRM ;

  file [SYSEXE]UCX$ENCODE.COM ;
  file [SYSEXE]UCX$DECODE.COM ;
  file [SYSEXE]UCX$RLOGIN.EXE ;
  file [SYSEXE]UCX$RSH.EXE ;
  file [SYSEXE]UCX$SMTP_RECEIVER.EXE ;
  file [SYSEXE]UCX$SMTP_SYMBIONT.EXE ;
  file [SYSEXE]UCX$UUENCODE.EXE ;
  file [SYSEXE]UCX$UUDECODE.EXE ;
  file [SYSEXE]UCX$FTPD.EXE ;
  file [SYSEXE]UCX$FTPC.EXE;
  file [SYSEXE]UCX$FTP.EXE ;
  file [SYSEXE]UCX$TELNET.EXE ;
  file [SYSEXE]TNDRIVER.EXE ;
  file [SYSEXE]UCX$LPD_SMB.EXE ;
  file [SYSEXE]UCX$LPQ.EXE ;
  file [SYSEXE]UCX$LPRM.EXE ;
  file [SYSEXE]UCX$LPD_RCV.EXE ;
  file [SYSEXE]UCX$LPRSETUP.EXE ;

  file [SYSHLP]UCX$FTP_HELP.HLB ;
  file [SYSHLP]UCX$TELNET_HELP.HLB ;

  file [SYSLIB]UCX$SMTP_MAILSHR.EXE ;
  file [SYSLIB]UCX$LPD_SHR.EXE ;
```

(continued on next page)

Example 2–2 (Cont.) UCX Product Description File

```
end option ;
end product ; ⓱
```

The following list describes the function of the PDL statements in the UCX example:

❶ The first line of the PDF is a comment line (identified by two hyphens).

❷ The *product* statement identifies the product as DEC VMS UCX V2.0. The full option specifies that the kit is a complete software distribution and not an update or patch.

❸ The *software* statement specifies that to run UCX, the execution environment must be running at least VMS Version 5.4.

❹ The *process parameter* statements specify the process parameters that the product requires. The utility will display a message about these requirements to users when they install UCX.

❺ The *rights identifier* statement causes the utility to display a message about a required rights identifier.

❻ The first *execute* statement specifies command procedures for product startup and shutdown.

❼ The second *execute* statement specifies the functional test for the product.

❽ The first *information* statement displays a message about preinstallation tasks. The utility prompts the user for confirmation of these tasks.

❾ The second *information* statement displays a message about postinstallation tasks.

❿ The *directory* statements create the directories [SYSTEST.UCX] and [SYSHLP.EXAMPLES.UCX].

⓫ The first *module* statement installs the help module for UCX in the default help library [SYSHLP]HELPLIB.HLB.

⓬ The second *module* statement installs the command module for UCX in the default command library [SYSLIB]DCLTABLES.EXE.

⓭ The next *file* statements install several required files for UCX.

⓮ The EXAMPLES option group contains programming example files that the user can select.

⓯ The NFS option group contains NFS protocol support files that the user can select.

⓰ The APPLICATION option group contains optional application support files and modules that the user can select.

⓱ The *end product* statement closes the group of *product* statements.

2.5 Writing a Platform PDF

As mentioned in Section 1.2, the POLYCENTER Software Installation utility gives you the means to create a software kit that is a combination of products (an integrated platform). To do this, you must create a platform PDF.

A platform PDF is distinguished from other types of PDFs by using the **platform** keyword to the *product* statement. In a platform PDF, you can use any statements to express the execution environment for the platform. For example, you can use the component option of the *software* statement to specify the software products that make up the platform.

When you package a platform, the PDFs and PTFs of the referenced products do not need to be present. However, when you copy a platform, products that are referenced with the component option of the *software* statement must be present.

The following example shows a platform PDF:

```
$ TYPE WORK3:[PCSI.KITS]OFFICE_PLAT.TXT
product DEC VAXVMS OFFICE_PLAT V1.0 platform ;
    file [SYSEXE]SOFTWARE.TST ;
    software DEC VAXVMS VMS
             version minimum V5.5 ;
    software DEC VAXVMS PROD_B
             version required 4.3 component ;
    software DEC VAXVMS PROD_C component ;
end product ;
```

The PDF in this example is for a platform named OFFICE_PLAT. The software products PROD_C and PROD_B are components of the platform. VMS Version 5.5 or greater must be present along with Version 4.3 of PROD_B.

2.6 Writing a Transition PDF

If you install a product using a mechanism other than the POLYCENTER Software Installation utility, the product database does not store information about the product. A **transition product description file** allows you to register a product in the product database even if it was installed using a mechanism other than the POLYCENTER Software Installation utility.

Once users register a product using the POLYCENTER Software Installation utility, they can perform utility operations on that product (for example, reconfigure or remove operations). Registering a product also allows you to use *software* statements or functions to reference that product. For more information about registering a product, see the *POLYCENTER Software Installation Utility User's Guide*.

A transition PDF must use the **transition** keyword to the *product* statement. The utility converts the transition PDF to the kit type of 6 and the file type PCSI$DESCRIPTION when you package it.

In a transition PDF, the *infer* statement is useful for testing the target system to determine if a product or product version is available.

The following example shows a transition PDF for the FMS product:

```
product DEC AXPVMS FMS V2.4 transition ; ❶
    infer version from [SYSLIB]FDVSHR.EXE ; ❷
    file [SYSLIB]FDVSHARE.OPT ; ❸
    module [SYSUPD]FDV.OBJ type object module FDV ; ❹
    module [SYSUPD]FDVMSG.OBJ type object module FDVMSG ;
    module [SYSUPD]FDVDAT.OBJ type object module FDVDAT ;
    module [SYSUPD]FDVERR.OBJ type object module FDVERR ;
    module [SYSUPD]FDVTIO.OBJ type object module FDVTIO ;
    module [SYSUPD]FDVXFR.OBJ type object module FDVXFR ;
    module [SYSUPD]HLL.OBJ type object module HLL ;
    module [SYSUPD]HLLDFN.OBJ type object module HLLDFN ;
end product ;
```

The following list describes the statements in this example:

❶ The **transition** keyword to the *product* statement indicates that this is a transition PDF.

❷ The *infer version* statement tests the execution environment to determine whether the file FDVSHR.EXE is present. If it is, the utility infers the version that is installed.

❸ The *file* statement indicates that the [SYSLIB]FDVSHARE.OPT file is part of the FMS kit.

❹ The *module* statements describe object modules in the default object library [SYSLIB]STARLET.OLB that are part of the FMS kit.

3

Creating the Product Text File

If you want to use PDF statements that display text to users, you must supply a product text file (PTF) as part of the product distribution. The PTF contains all the text that you want to display to users when they perform POLYCENTER Software Installation utility operations. You supply this text file, along with the PDF and product material, during a package operation. By providing adequate help text in the PTF, you should be able to reduce or eliminate hardcopy installation documentation.

The following PDF statements have corresponding text in the PTF:

- *error*
- *information*
- *option*
- *part*
- *product*

3.1 PTF File Name Format

The PTF must reside in the same directory as the PDF. It must also have the same file name as the PDF and a file type of .PCSI$TEXT.

The following are examples of valid input PTF and PDF names:

```
TEST-INSTALL.TXT
TEST-INSTALL.PCSI$TEXT
BARRE-BLACKJACK-V2.PDF
BARRE-BLACKJACK-V2.PCSI$TEXT
```

3.2 PTF Format

The **standard format** for the PTF is a text file (containing ISO Latin-1 characters) that you create with a text editor. Because the PTF is a text file and separate from the PDF, it has the advantage of being platform independent. Note that there are no comment characters in PTFs.

The utility converts your standard format PTF to the **reference format** during a package operation. Converting the output PTF to reference format improves performance. (Note that the reference format of the PTF is unrelated to the concept of a reference copy, which was discussed in Chapter 1.)

3.2.1 Specifying the Product Name

You must use the =product directive to specify product information in the PTF. The information that you specify with the =product directive must match the information you specify with the *product* statement in the PDF.

The =product directive has the following format:

=product *producer ABI product version kit_type*

where:

- *producer* is the legal owner of the software product. For example, for Digital layered products, this part of the =product directive is DEC.
- *ABI* is the Application Binary Interface (ABI), which specifies the hardware and software that the product runs on. For example, layered products that execute on OpenVMS VAX systems use VAXVMS in this part of the =product directive. OpenVMS layered products that execute on OpenVMS AXP systems use AXPVMS in this part of the =product directive.
- *product* is the name of the software product. The combination of *producer*, *ABI*, and *product* must be unique.
- *version* is the version identifier of the product in the following format:

  type-version-update-edit-patch

  where:

  - *type* is a single uppercase letter (for example, V).
  - *version* is a decimal integer from 0 through 99 that identifies a major software release.
  - *update* is a decimal integer from 0 through 99 that identifies a minor release.
  - *edit* is an optional positive decimal integer you can use to indicate the edit level of software. Separate the edit level from the rest of the version identifier with a hyphen (-). The hyphen is required *even if you do not specify an edit level.*
  - *patch* is any string other than a postitive decimal integer (optional).
- *kit_type* is a keyword that specifies a PDF type. Use the same keywords as those listed for the **type** parameter of the *product* statement.

For example:

```
=product DEC AXPVMS FMS V2.4 full
```

This indicates that the product is Digital VMS FMS Version 2.4.

3.2.2 PTF Modules

PTF text modules are text blocks that you want to present to the user. The POLYCENTER Software Installation utility does not process text blocks sequentially, so the order of the text modules in the PTF does not matter.

Text modules are identified by a module header line in the following format:

1 module-name

The module header line consists of the number 1, followed by a space or tab and the name of the module. The *module-name* must be from 1 to 31 ISO Latin-1 characters, excluding the horizontal tab, space, exclamation point (!), and comma (,) characters. For example:

```
1 SAMPLE
```

The POLYCENTER Software Installation utility uses the name of the module to associate the text module with a line from the PDF. For example, the SAMPLE module could correspond to an option in the PDF:

```
option SAMPLE ;
```

When a user chooses the SAMPLE option, the utility displays the help text for the choice. Note that the format of the help text in the PTF (for example, spacing and blank lines) is preserved when the utility displays it to the user.

The utility also allows you to specify text modules that are not associated with statements in the PDF. These text modules are preceded by an apostrophe ('). Use the following module names to specify information about your product:

- The 'LICENSE module specifies licensing information.
- The 'NOTICE module specifies copyright, ownership, and similar legal information.
- The 'PRODUCER module specifies a brief description of the producer of the product.
- The 'PRODUCT module specifies a brief functional description of the product.

For example, a product might contain the following modules:

```
=product DEC VAXVMS C V1.0 full
1 'PRODUCT
=prompt DEC C++ for OpenVMS
DEC C++ for OpenVMS VAX is a native compiler that implements the C++
programming language and includes:

o  A C++ compiler that implements C++ as defined by The Annotated C++
   Reference Manual, Ellis & Stroustrup, reprinted with corrections,
   May 1991. The compiler implementation includes templates but ex-
   cludes exception handling. DEC C++ generates optimized object code
   without employing an intermediate translation to C.

o  The DEC C++ Class Library, which consists of the following class li-
   brary packages: iostream, complex, generic, Objection, Stopwatch,
   String, task, messages, and vector.
1 'NOTICE
=prompt © Digital Equipment Corporation 1992. All rights reserved.
Unpublished rights reserved under the copyright laws of the United States.

This software is proprietary to and embodies the confidential technology of
Digital Equipment Corporation. Possession, use, or copying of this software
and media is authorized only pursuant to a valid written license from Digital
or an authorized sublicensor.
```

```
Restricted Rights:  Use, duplication, or  disclosure by the U.S.
Government is subject to restrictions as set forth in subparagraph (c)(1)(ii)
of DFARS 252.227-7013, or in FAR 52.227-19, or in FAR 52.227-14 Alt. III,
as applicable.
1 'LICENSE
=prompt This product uses the PAKs: <xxx> and <xxx-RT>.
This software is furnished under the licensing provisions of Digital
  Equipment Corporation's Standard Terms and Conditions. For more in-
  formation about Digital's licensing terms and policies, contact your
  local Digital office.
1 'PRODUCER
=prompt Digital Equipment Corporation
This software product is sold by Digital Equipment Corporation.
```

To see how the POLYCENTER Software Installation utility displays this text, see the *POLYCENTER Software Installation Utility User's Guide*.

3.2.3 Including Prompt and Help Text

You can include prompt and help text in your PTF using the =prompt directive. Prompt text cannot exceed one line of text. Help text is similar to prompt text, except that it can span multiple lines. The help text follows the =prompt line. You can also include blank lines in help text.

The following example shows prompt text:

```
=prompt This option provides files for programming support.
```

The following example shows the product text file associated with TCP/IP Services for OpenVMS. Note the prompt and help text:

```
=product DEC VAXVMS UCX V2.0 full
1 'PRODUCT
=prompt DEC TCP/IP Services for OpenVMS
 DEC TCP/IP Services for OpenVMS is an OpenVMS layered software product that
 promotes interoperability and resource sharing between OpenVMS systems,
 UNIX systems, and other systems that support the TCP/IP and NFS
 protocol suites.

 The product provides capabilities for file access, remote terminal
 access, remote command execution, remote printing, mail, and application
 development, including three major functional components:

 o  The Run-Time component, which is based on the Berkeley Standard
    Distribution, brings TCP/IP communications to OpenVMS computer systems with
    the following TCP/IP protocols and features:  Transmission Control
    Protocol (TCP), Internet Protocol (IP), Internet Control Message
    Protocol (ICMP), Address Resolution Protocol (ARP), User Datagram
    Protocol (UDP), Routing Information Protocol (RIP), Berkeley Internet
    Name Domain (BIND) Resolver, Simple Network Management Protocol (SNMP)
    Agent, Ethernet support, FDDI support, DEC TCP/IP Auxiliary server
    (inetd), DEC TCP/IP system management, and Security and network access
    control.

    The Run-Time component includes a suite of application development tools
    (DECrpc, C socket programming interface, and QIO programming
    interface).

    DECrpc lets users take advantage of all the available computing
    cycles on the network. DECrpc is a port of the Hewlett-Packard Network
    Computing Systems. It provides a set of tools
    for building multivendor, distributed applications. This lets
    programmers distribute applications across multiple network nodes,
    taking advantage of each system's capabilities and enhancing the
    network's overall performance. DECrpc includes the following components:
    Remote Procedure Call (RPC) Run-Time Library, Network Interface
    Definition Language (NIDL) Compiler, and a Location Broker.
```

```
    DECrpc does not include other components of the Network Computing
    Architecture, such as the Concurrent Programming Support or the
    replicated Global Location Broker.

 o  The Applications component includes the popular user-oriented protocols
    for file transfer, remote processing, remote printing, and mail: File
    Transfer Protocol (FTP), Telnet Protocol (Telnet), Berkeley R commands
    (rsh, rlogin, rexec), remote printing, and Simple Mail Transfer
    Protocol (SMTP).

 o  The DEC NFS component supports Network File System (NFS) V2.0 proto-
    col specifications. NFS is an Application layer protocol that provides
    clients with transparent access to remote file services. The DEC NFS
    implementation is unique in the industry in that it provides an NFS
    cluster alias failover capability as well as access to both an OpenVMS and
    ULTRIX compatible file system. With these features, UNIX systems can
    share files between an OpenVMS file system and an ULTRIX file system and
    more fully exploit the reliability of VAXcluster systems. DEC NFS does
    not support OpenVMS clients.

    The DEC NFS server promotes data sharing among clients by providing
    a central data storage facility for OpenVMS and UNIX files. The DEC NFS
    server provides two types of file access for UNIX clients: 1) client
    access to OpenVMS files, and 2) client access to files compatible with
    UNIX systems.
1 'NOTICE
=prompt © Digital Equipment Corporation 1992. All rights reserved.
Unpublished rights reserved under the copyright laws of
the United States.

This software is proprietary to and embodies the confidential technology of
Digital Equipment Corporation. Possession, use, or copying of this software
and media is authorized only pursuant to a valid written license from Digital
or an authorized sublicensor.

Restricted Rights:  Use, duplication, or  disclosure by the U.S.
Government is subject to restrictions as set forth in subparagraph (c)(1)(ii)
of DFARS 252.227-7013, or in FAR 52.227-19 or in FAR 52.227-14 Alt. III, as
applicable.
1 'LICENSE
=prompt This product uses the PAKs: UCX and UCX-IP-RT.
This product currently has two Product Authorization Keys (PAKs):

   Producer  PAK Name  Version  Release Date

     DEC     UCX         2.0     6-JUL-1992
     DEC     UCX-IP-RT   2.0     6-JUL-1992
1 'PRODUCER
=prompt Digital Equipment Corporation
This software product is sold by Digital Equipment Corporation.
1 EXAMPLES
=prompt Example files
The example files include client/server programming examples.
1 NFS
=prompt NFS files
The DEC NFS component supports Network File System (NFS) protocol
specifications. NFS is an Application layer protocol that provides
clients with transparent access to remote file services. The DEC NFS
implementation is unique in the industry in that it provides an NFS
cluster alias failover capability as well as access to both an OpenVMS and
ULTRIX compatible file system. With these features, UNIX systems can
share files between an OpenVMS file system and an ULTRIX file system and
more fully exploit the reliability of VAXcluster systems. DEC NFS does
not support OpenVMS clients.
```

```
The DEC NFS server promotes data sharing among clients by providing
a central data storage facility for OpenVMS and UNIX files. The DEC NFS
server provides two types of file access for UNIX clients: 1) client
access to OpenVMS files, and 2) client access to files compatible with UNIX
systems.
1 APPLICATIONS
=prompt Applications
The Applications component includes the popular user-oriented protocols
for file transfer, remote processing, remote printing, and mail: File
Transfer Protocol (FTP), Telnet Protocol (Telnet), Berkeley R commands
(rsh, rlogin, rexec), remote printing, and Simple Mail Transfer
Protocol (SMTP).
1 PRE_INSTALL
=prompt Complete preinstallation tasks for DEC TCP/IP Services first.
Before you install DEC VMS UCX, you must complete certain preinstallation
tasks. For more information, refer to the "DEC TCP/IP Services for VMS
Installation and Configuration Guide."
1 POST_INSTALL
=prompt Postinstallation tasks required for DEC TCP/IP Services.
For more information, refer to these associated documents:

- "DEC TCP/IP Services for VMS Installation and Configuration Guide"
- "DEC TCP/IP Services for VMS System Management"
```

4

Packaging the Kit

When you package your product, it takes one of the following forms:

- **Reference copy**. This is the standard form your product takes as a result of a package operation. It is ready for installation and is in a directory tree on a random-access device (for example, a compact disc drive).
- **Sequential copy**. This an optional form your product takes as a result of a package operation. It consists of a set of container files that can be placed on a sequential-access device (for example, a magnetic tape drive). The POLYCENTER Software Installation utility converts the product to reference format during an installation.

This chapter describes how to create a reference or sequential copy. It also describes how to copy the kit.

Note

If you are creating a reference copy, do not use the same directory as the source for your product material and the destination for the reference copy.

4.1 Creating Reference and Sequential Copies

To convert your product to a reference or sequential copy, perform a package operation, as described in the following sections.

4.1.1 Packaging with the DECwindows Motif Interface

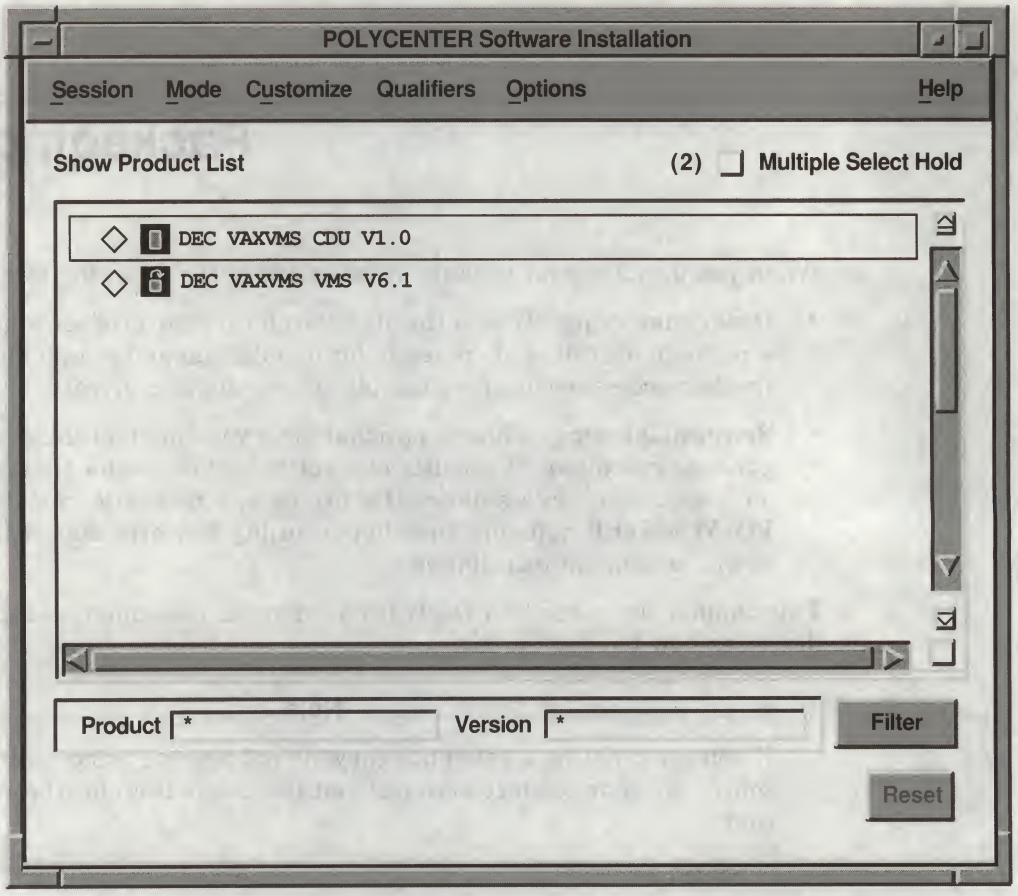
To package your product using the DECwindows Motif interface, do the following:

1. Start the DECwindows Motif interface to the POLYCENTER Software Installation utility by entering the SET DISPLAY and PRODUCT commands. For example:

   ```
   $ SET DISPLAY/CREATE/NODE=MYNODE
   $ PRODUCT
   ```

   The POLYCENTER Software Installation utility displays the main window as shown in Figure 4–1.
2. Choose Package from the Mode menu.
3. If the source for your package operation is not defined, the utility displays an error message, which you can dismiss.

Figure 4–1 POLYCENTER Software Installation Utility Main Window



ZK–6268A–GE

4. Enter the search path to your PDF and PTF in the Source field, and click on Filter.
5. The product you want to package will appear in the Package Kit List.
6. Double click on the product you want to package (in the Package Kit List). The product moves from the Package Kit List to the Selected Kits to be Packaged window.
7. Enter the search path to your product material in the Material field.
8. Enter the disk and directory where you want to place the files in the Destination field.
9. By default, the output format is the same as the input format. To explicitly create a reference or sequential copy, click on the Package Format field.
10. Click on the Package button to begin the package operation. A Work-in-Progress dialog box displays information about the status of the package operation.
11. To save current settings such as the source, destination, and material location, choose Save settings on the Customize menu. The utility displays the settings you saved each time you start the utility and select Package mode.

4.1.2 Packaging with the DCL Interface

You can enter the DIGITAL Command Language (DCL) command PRODUCT PACKAGE in the following format:

```
PRODUCT PACKAGE product_name /SOURCE=search_path -
/DESTINATION=directory /MATERIAL=search_path
```

where *product_name* specifies the name of the software you want to package. Note that you cannot specify hyphens (-) in the *product_name*. Depending on the operation you are performing, you must specify some of the qualifiers in Table 4–1.

Table 4–1 PRODUCT PACKAGE Command Qualifiers

Qualifier	Meaning
/PRODUCER=*name*	Specifies the name of the company that manufactures the software. If you omit the /PRODUCER qualifier, the default is the wildcard character (*).
/BASE_SYSTEM=*name*	Specifies the name of the hardware and operating system environment you want to use with the software. (This is sometimes referred to as the platform.) If you omit the /BASE_SYSTEM qualifier, the default is the environment from which you are running the POLYCENTER Software Installation utility.
/COPY	Specifies whether you want the product material copied into the reference copy. /COPY is the default. Specifying the /NOCOPY qualifier can save processing time when you are debugging a PDF.
/DESTINATION=*directory*	Specifies the primary disk and directory for placement of software product files. You must specify this qualifier with package and copy operations.
/FORMAT	Specifies that the output format of the product kit being packaged or copied is either sequential (use /FORMAT=SEQUENTIAL) or reference (use /FORMAT=REFERENCE). The reference format is the default for a package operation. In a copy operation, the output format is the same as the input format.
/LOG	Monitors the activity of the POLYCENTER Software Installation utility as it performs an operation. Messages are sent to SYS$OUTPUT (usually defined as your terminal). By default, no activity logging occurs.
/MATERIAL=*search_path*	Determines the source of product material. There is no default for package operations; you must specify this qualifier.
/OWNER_UIC=*uic*	Specifies the owner user identification code (UIC) for files created during a copy or package operation. By default, the files for a software product are owned by the user executing the operation. For example, if you are logged in to your own account, you can use this qualifier during a package operation to assign ownership of the product files to SYSTEM rather than to your own account.

(continued on next page)

Table 4–1 (Cont.) PRODUCT PACKAGE Command Qualifiers

Qualifier	Meaning
/SOURCE=*search_path*	Specifies the location of the software product that you want to install or, in a package or copy operation, the location of your PDF and PTF. Note that during a copy or package operation, the POLYCENTER Software Installation utility renames the PDF in the output kit. For more, information see Section 2.3.
/TRACE	Traces the activity of the POLYCENTER Software Installation utility to assist in development of a product description file (PDF). /TRACE provides more information than /LOG. By default, no activity tracing occurs.
/VERSION=*string*	Indicates the version number of the software that you want to use. If you do not specify the /VERSION qualifier, the default is all versions found. If the utility finds more than one version, you are asked to choose which one you want. Note that by default the utility looks only for a version number that begins with a V, although you can use the /VERSION qualifier to specify a version that begins with another character, such as one you might use for a field test version.

Example

The following command packages VAX RALLY and creates a reference copy in the [SOFTWARE_KITS] directory:

```
$ PRODUCT PACKAGE RALLY /LOG /SOURCE=[ZABREWSKI.RALLY] -
_$ /DESTINATION=[SOFTWARE_KITS] /MATERIAL=[ZABREWSKI...]

The following product has been selected:
DEC VAXVMS RALLY V3.4

Do you want to continue [YES]
%PCSI-I-SUCCESS, operation completed successfully
$
```

Because this example does not specify product information, the utility uses the default for the producer, base system, and version.

4.2 Copying Kits

You can make copies of kits using the copy operation, as described in the following sections.

4.2.1 Copying Kits with the DECwindows Motif Interface

To copy your kit using the DECwindows Motif interface to the POLYCENTER Software Installation utility, do the following:

1. Start the DECwindows Motif interface to the utility by entering the SET DISPLAY and PRODUCT commands. For example:

   ```
   $ SET DISPLAY/CREATE/NODE=MYNODE
   $ PRODUCT
   ```

   The POLYCENTER Software Installation utility displays the main window as shown in Figure 4–1.

2. Choose Copy from the Mode menu.

3. If the source for your copy operation is not defined, the utility displays an error message, which you can dismiss.
4. Enter the search path to your PDF and PTF in the Source field, and click on Filter.
5. The product you want to copy appears in the Copy Kit List.
6. Double click on the product you want to Copy (in the Copy Kit List). The product moves from the Copy Kit List to the Selected Kits to be Copied window.
7. Enter the disk and directory where you want to place the copy in the Destination field.
8. By default, the utility creates a reference copy. To create a sequential copy, click on the Copy Format field.
9. Click on the Copy button to begin the copy operation. A Work-in-Progress dialog box displays information about the status of the copy operation.
10. To save current settings such as the source and destination, choose Save settings on the Customize menu. The utility displays the settings you saved each time you start the utility and select copy mode.

4.2.2 Copying Kits with the DCL Interface

To copy your kit using the DCL interface to the POLYCENTER Software Installation utility, use the PRODUCT COPY command. If you do not use the /FORMAT qualifier, the utility creates a copy in the same format as the original kit.

To copy a kit to a different format (for example, from reference to sequential), use the /FORMAT qualifier. You must also specify the /SOURCE and /DESTINATION qualifiers. To create a sequential copy (from a reference copy), use the following format:

```
PRODUCT COPY/FORMAT=SEQUENTIAL product_name -
/SOURCE=filename /DESTINATION=filename
```

where *product_name* is the name of the software. Note that you cannot specify hyphens (-) in the *product_name*. For an explanation of the qualifiers, see Table 4–1.

To create a reference copy (from a sequential copy), replace the /FORMAT=SEQUENTIAL qualifier with the /FORMAT=REFERENCE qualifier.

Examples

1. The following command converts OpenVMS CMS to a sequential copy in the [KIT.SEQUENTIAL] directory:

```
$ PRODUCT COPY CMS /FORMAT=SEQUENTIAL /SOURCE=[MILLER.CMS] -
_$ /DESTINATION=[KIT.SEQUENTIAL]

The following product has been selected:
DEC VAXVMS CMS V3.4

Do you want to continue [YES]

Percent Done: 10%...30%...60%...90%...100%
%PCSI-I-SUCCESS, operation completed successfully
$
```

   Because this example does not specify product information, the utility uses the default for the producer, base system, and version.

2. The following command converts a VAX Ada sequential copy to a reference copy in the [KITS.FINAL] directory:

```
$ PRODUCT COPY ADA /FORMAT=REFERENCE /SOURCE=[MILLER.ADA] -
_$ /DESTINATION=[KITS.FINAL]

The following product has been selected:
DEC VAXVMS ADA V4.1

Do you want to continue [YES]

Percent Done: 10%...20%...40%...60%...80%...90%...100%
%PCSI-I-SUCCESS, operation completed successfully
$
```

Because this example does not specify product information, the utility uses the default for the producer, base system, and version.

Part II

Product Description Language Statements

Part II contains descriptions of individual product description language statements.

account

The *account* statement uses a command procedure to create a system account.

Syntax

account *name* with *(parameters,...)* ;

Parameters

name
Specifies the user name of the account. The user name is passed to the command procedure as P1.

with *(parameters,...)*
Specifies the list of parameters that are passed to the command procedure that creates the account. Each parameter must be a single unquoted or quoted string that specifies P2 through P8, in order. Refer to the Description section for the meaning of the parameters.

Description

The *account* statement uses a command procedure (SYS$UPDATE:PCSI$CREATE_ACCOUNT.COM) to create an account. The parameters that you pass to the command procedure that creates the accounts are

- P1 specifies the user name of the account (using the **name** parameter).
- P2 specifies general AUTHORIZE qualifiers.
- P3 specifies a comma-separated list of rights identifiers to grant to the user name.
- P4 through P8 specify other general AUTHORIZE qualifiers.

When you remove a product that created accounts, the POLYCENTER Software Installation utility uses a command procedure (SYS$UPDATE:PCSI$DELETE_ACCOUNT.COM) to delete accounts associated with your product.

The *account* statement specifies an account managed object that has the following characteristics:

- Its name is the value of the **name** parameter. The name must be unique among all account names in the operating scope.
- It has operating lifetime and operating scope.
- Managed object conflict is not recoverable.

See Also

rights identifier

Example

```
account test with ("/priv=tmpmbx,netmbx");
```

In this example, the *account* statement creates the TEST account. The command procedure that creates the account also assigns the TMPMBX and NETMBX privileges to the account.

apply to

The *apply to* statement specifies a product that you want to update with a mandatory update or patch product description.

You must include an *apply to* statement in a mandatory update or patch PDF to identify the product that is being patched or updated. This statement is not valid in other types of PDFs.

Syntax

apply to *producer abi name* { [version below *version* / version maximum *version* / version minimum *version*] / version required *version* } ;

Parameters

producer
Specifies the legal owner of the software product.

abi
Specifies the Application Binary Interface (ABI) on which the product executes.

name
Specifies the product name. The combination of producer name and product name must be unique.

Options

version below *version*
Specifies the smallest invalid product version. Use this option to specify that the product version must be less than (but not equal to) the specified version. You cannot use this option with either the version minimum or version required options. By default, there is no smallest invalid version.

version maximum *version*
Specifies a maximum product version that must be available. Use this option to specify that the product version must be less than or equal to the specified version. You cannot use this option with the version below option. By default, there is no maximum version.

version minimum *version*
Specifies a minimum product version that must be available. Use this option to specify that the product version must be greater than or equal to the specified version. By default, there is no minimum version.

version required *version*
Specifies a required product version that must be available. Use this option to specify that a specific product version must be present. You cannot use this option with either the version below, version maximum, or version minimum options. By default, there is no required version.

apply to

Description

The *apply to* statement specifies which versions of another product must be available for a valid installation operation. You can use options to this statement to define minimum, maximum, and required versions.

If you use the version minimum or version maximum option, the POLYCENTER Software Installation utility searches in the following order the first character of the version identifier of available products for a match:

V
T
E
X
S
D
B
null

If the utility finds a match, it compares the rest of the version identifier to the constraints you specified with the version minimum or version maximum option.

If your product references another product with an *apply to* statement, the referenced product will be installed earlier than, and removed later than, your product. If two products reference each other (creating an infinite loop), the utility issues an error message.

The *apply to* statement is a utility directive and does not specify a managed object.

See Also

product
upgrade
software

Example

```
product DEC VAXVMS CSCPAT57 V1.0 patch ;
    apply to DEC VAXVMS FORTRAN version required V2.0 ;
    patch image [SYSEXE]FORTRAN.EXE with [000000]CSCPAT57.PAT ;
end product ;
```

This example shows part of the product description for a patch to DEC Fortran. As shown in the *apply to* statement, you must be running DEC Fortran Version 2.0 to apply this patch.

bootstrap block

The *bootstrap block* statement specifies the file that the bootstrap block references.

Syntax

bootstrap block *name* image *source* ;

Parameters

name
Specifies the bootstrap file specification. You must define the file you specify in the same product description (with a *file* statement). You must also ensure that the file has bootstrap scope and product or assembly lifetime (using the *scope* statement).

image *source*
Specifies the file specification of the file that contains the bootstrap block image. The referenced file must be defined in the same product description (with a *file* statement), and it must also have product scope and product lifetime.

Description

The *bootstrap block* statement specifies the file that the bootstrap block references. You specify the name of the file as the **name** parameter.

The *bootstrap block* statement also specifies a bootstrap block managed object that has the following characteristics:

- It is unnamed and unique within the bootstrap scope.
- It has operating lifetime and bootstrap scope.
- Managed object conflict is not recoverable.

See Also

scope

Example

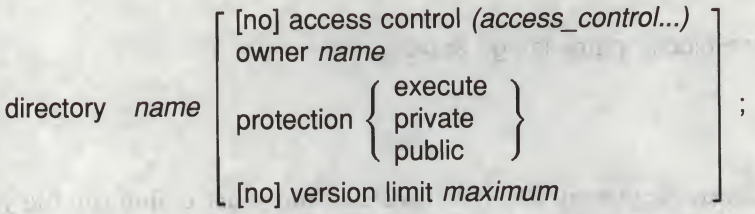
```
bootstrap block [sysexe]vmb.exe image [sysexe]bootblock.exe ;
```

This example uses the *bootstrap block* statement to point the bootstrap block to the bootstrap file ([SYSEXE]VMB.EXE).

directory

The *directory* statement creates the specified directory if it does not already exist.

Syntax



Parameter

name
Specifies the directory name.

Options

[no] access control *(access_control...)*
Specifies the minimum access control entries (ACEs) that the directory will have. You must specify the ACEs as a quoted string. By default, directories have no added ACEs.

owner *name*
Specifies the account name that owns the directory. By default, the directory is owned by the SYSTEM account.

protection execute
Sets the directory protection so that users have execute access.

protection private
Sets the directory protection so that users have no access.

protection public
Sets the directory protection so that users have read and execute access. This is the default option.

[no] version limit *maximum*
Specifies the maximum number of file versions in the directory as an unsigned integer from 1 through 32767.

The default is no version limit.

Description

The *directory* statement creates the specified directory if it does not already exist.

The *directory* statement specifies the name of a directory managed object. Check the other statements in your PDF to make sure the name you specify is unique among all directory, file, and link managed objects in all scopes.

The scope and lifetime of the directory managed object depend on whether it is lexically contained in a scope group, as shown in Table PDF–1.

Table PDF–1 Directory Managed Object Scope and Lifetime

Type of Scope Group	Lifetime	Scope
Product	Product	Product
Global	Assembly	Global
Bootstrap	Operating	Bootstrap
Operating	Operating	Operating
Processor	Operating	Processor

If you use the access control option, the *directory* statement specifies one access control entry (ACE) managed object that references the directory managed object for each entry specified with the access control option. The ACE managed object has the following characteristics:

- It is unnamed.
- It has operating lifetime.
- It has the same scope as the directory.

See Also

scope

Example

```
directory [SYSHLP.EXAMPLES.FMS.MESSAGE] protection private ;
```

This example specifies the directory [SYSHLP.EXAMPLES.FMS.MESSAGE]. The protection private option specifies that no users have access to this directory.

end

Ends a group of statements. For more information, see the corresponding opening statement.

Syntax

```
end if ;
end option ;
end part ;
end product ;
end remove ;
end scope ;
```

error

The *error* statement displays an error message during an installation or reconfiguration operation. The text is from a PTF text module.

Syntax

error *name ;*

Parameter

name
Specifies, as a quoted or unquoted string, the name of the associated PTF text module. The name you specify must be unique among all names in the same product description.

Description

The *error* statement specifies a text module you want to display during an installation or reconfiguration operation. The *error* statement must be immediately contained within an if group.

The POLYCENTER Software Installation utility processes *error* statements in lexical order. The utility displays prompt and help text during the validation phase (which follows the configuration phase).

After receiving an error, the utility prompts the user to continue or terminate the operation. If the operation is not executed interactively, it terminates. Each *error* statement results in a separate prompt.

The *error* statement is a utility directive and does not specify a managed object.

You must supply text in the associated product text module. The module must contain a =prompt directive line.

See Also

if

Example

Suppose the product description file contains the following lines:

```
if (<hardware processor model 7>) ;
    error UNSPROC ;
end if ;
```

The corresponding module in the PTF could contain the following lines:

```
1 UNSPROC
=prompt Not supported on MicroVAX I.
This product is not supported on the MicroVAX I processor.
```

If the processor model is 7, the system displays the following message:

```
Not supported on MicroVAX I.

This product is not supported on the MicroVAX I processor.
Do you want to abort [YES]
```

execute install, remove

The *execute install* statement specifies commands that you want to execute when the product is installed. The *remove* part of the command indicates commands you want to execute when the product is removed from the execution environment. The POLYCENTER Software Installation utility cannot execute these commands because they are specific to a product and outside the management domain of the utility.

Syntax

execute install *(command,...)* remove (command,...) [uses *(file,...)*] ;

Parameter

(command,...)
Specifies the command that the POLYCENTER Software Installation utility passes to the command interpreter in the execution environment.

Option

uses *(file,...)*
Specifies the files required to execute the commands you specified in the **command** parameter. Specify the list of required files with the uses option. By default, this statement does not require files.

Description

The *execute install, remove* statement specifies commands that you want to execute when the product is installed. The *remove* part of the command indicates commands that execute when the product is removed from the execution environment.

The POLYCENTER Software Installation utility cannot execute these commands because they are specific to a product and outside the management domain of the utility. You specify actions by entering a command line, which the utility passes to a command interpreter in the execution environment.

The *scope* statement controls the execution of the commands; the commands execute once in each scope.

If you need files for the *execute install, remove* statement, specify the uses option. Each file you specify with the uses option must be present in the product material.

The *execute install* statement causes the system to define several logical names. The commands must use these logical names to reference files, as follows:

- PCSI$SOURCE is a root directory specification that points to the reference copy.
- PCSI$DESTINATION is a root directory specification that points to the root directory for the current scope.
- PCSI$WORK is a subdirectory under [SYSUPD] that can contain temporary files.

execute install, remove

The *execute install* statement is a utility directive and does not specify a managed object.

See Also

file (assemble execute option)
file (release execute option)

Example

```
execute
   install "@pcsi$root:[sysupd]load_loadable_image.com"
   remove "@pcsi$root:[sysupd]unload_loadable_image.com"
   uses ([sysupd]load_loadable_image.com,
         [sysupd]unload_loadable_image.com) ;
```

In this example, the *execute install, remove* statement sets up command procedures to run when the product is installed and removed. The uses option specifies the file names of the command procedures for installation and removal.

execute login

The *execute login* statement displays a message to users that they should execute the specified commands when the product is made available to a process.

Syntax

execute login *(command,...)* ;

Parameter

(command,...)
Specifies the command that the POLYCENTER Software Installation utility passes to the command interpreter in the execution environment.

Description

The *execute login* statement displays a message to users that they should execute the specified commands when the product is made available to a process. You can use this statement to advise users of commands they should add to their login files.

The *execute login* statement does not specify a managed object.

See Also

file (assemble execute option)
file (release execute option)

Example

```
execute login "$ @USER_START" ;
```

In this example, the *execute login* statement displays a message to users about adding a line to their LOGIN.COM file.

execute postinstall

The *execute postinstall* statement specifies commands that execute after the product is made available to the system. The POLYCENTER Software Installation utility cannot execute these commands because they are specific to a product and outside the management domain of the utility.

Syntax

execute postinstall *(command,...)* [uses *(file,...)*] ;

Parameter

(command,...)
Specifies the command that the POLYCENTER Software Installation utility passes to the command interpreter in the execution environment.

Option

uses *(file,...)*
Specifies the files required to execute the commands you specified in the **command** parameter. Specify the list of required files with the uses option. By default, this statement does not require files.

Description

The *execute postinstall* statement specifies commands that execute after the product is made available to the system.

The POLYCENTER Software Installation utility cannot execute these commands because they are specific to a product and outside the management domain of the utility. You specify actions by entering a command line, which the utility passes to a command interpreter in the execution environment.

The *scope* statement controls the execution of the commands; the commands execute once in each scope.

If you need files for the *execute postinstall* statement, specify the uses option. Each file you specify with the uses option must be present in the product material.

The *execute postinstall* statement is a utility directive and does not specify a managed object.

See Also

file (assemble execute option)
file (release execute option)

Example

```
execute
    postinstall "@pcsi$root:[sysupd]product_cleanup.com"
    uses [sysupd]product_cleanup ;
```

In this example, the *execute postinstall* statement sets up a command procedure to run after the product is installed. The uses option specifies the file name of the command procedure.

execute release

The *execute release* statement specifies commands to execute when the existing product version is replaced with a later version.

Syntax

execute release *(command,...)* [uses *(file,...)*] ;

Parameter

(command,...)
Specifies the commands the POLYCENTER Software Installation utility passes to the command interpreter in the execution environment.

Option

uses *(file,...)*
Specifies the files required to execute the commands you specified in the **command** parameter. Specify the list of required files with the uses option. By default, this statement does not require files.

Description

The *execute release* statement specifies commands that execute when the existing product version is replaced with a later version.

The *execute release* statement causes the system to define several logical names. The commands must use these logical names to reference files, as follows:

- PCSI$SOURCE is a root directory specification that points to the reference copy.
- PCSI$DESTINATION is a root directory specification that points to the root directory for the current scope.
- PCSI$WORK is a subdirectory under [SYSUPD] that can contain temporary files.

The *execute release* statement is a utility directive and does not specify a managed object.

See Also

file (assemble execute option)
file (release execute option)

Example

```
execute release "@pcsi$source:[sysupd]config" uses [SYSUPD]CONFIG.COM ;
```

In this example, the *execute release* statement sets up a command procedure to run when the product is upgraded. The uses option specifies the file name of the command procedure.

execute start, stop

The *execute start* statement displays a message to users indicating commands they should execute when the product is made available on the processor to which it is bound. The *stop* part of the statement indicates commands that execute when the product is made unavailable on the processor to which it is bound.

Syntax

execute start *(command,...)* stop *(command,...)* ;

Parameter

(command,...)
Specifies the command that the POLYCENTER Software Installation utility passes to the command interpreter in the execution environment.

Description

The *execute start, stop* statement displays a message to users indicating commands they should execute when the product is made available on the processor to which it is bound. The *stop* part of the command indicates commands that execute when the product is made unavailable on the processor to which it is bound. (Note that the *stop* part of the command is required even if there are no commands that you want to execute when the product is made unavailable.) The POLYCENTER Software Installation utility cannot execute these commands because they are specific to a product and outside the management domain of the utility.

If you need files for the *execute start, stop* statement, you must provide them with the *file* statement.

The *execute start, stop* statement is a utility directive and does not specify a managed object.

See Also

file (assemble execute option)
file (release execute option)

Examples

1.
```
execute
    start "@sys$startup:product_startup.com"
    stop "@sys$startup:product_shutdown.com" ;
```

   In this example, the *execute start, stop* statement displays a message to users about command procedures they should run to start and stop the product.

2.
```
execute
    start "@sys$startup:abs_startup.com"
    stop "" ;
```

   In this example, the *execute start* statement displays a message to users about command procedures they should run to start the product. Note that there are no commands executed when the product is stopped.

execute test

The *execute test* statement specifies commands that execute during the functional test of the product.

Syntax

execute test *(command,...)* ;

Parameter

(command,...)
Specifies the command that the POLYCENTER Software Installation utility passes to the command interpreter in the execution environment.

Description

The *execute test* statement specifies commands that execute during the functional test of the product. The POLYCENTER Software Installation utility cannot execute these commands because they are specific to a product and outside the management domain of the utility. You specify actions by entering a command line, which the utility passes to the DIGITAL Command Language (DCL) command interpreter.

If you need files for the *execute test* statement, you must provide them with the *file* statement.

The *execute test* statement is a utility directive and does not specify a managed object.

See Also

file (assemble execute option)
file (release execute option)

Example

```
execute
    test "@sys$test:prod$ivp.com" ;
```

In this example, the *execute test* statement sets up a command procedure to run during the functional test of the product.

file

The *file* statement creates a file in the execution environment. If a file of the same name already exists, the POLYCENTER Software Installation utility might replace the file, depending on the options specified.

Syntax

file *name*

```
[ [no] access control (access_control...)
  [no] archive
  assemble { copy
           { execute (command,...) [uses (file,...)]
           { requires file (file,...)
  [no] generation generation
  image library
  image module module_name
  owner owner
  protection { execute
             { private
             { public
  release execute (command,...) [uses (file,...)]
  { release merge
  { release replace
  release notes
  size size
  source source
  [no] write ]
```

;

Parameter

name
Specifies the relative file specification of the file.

Options

[no] access control *(access_control...)*
Specifies the minimum access control entries (ACEs) that the file will have. By default, files have no added ACEs (no access control).

[no] archive
Allows you to preserve existing files during an upgrade. The utility appends *_OLD* to the end of the file type. For example, if you archived an existing file named STARTUP_TEMPLATE.SYS, the utility would rename it STARTUP_TEMPLATE.SYS_OLD. If there are several versions of the existing file, the utility purges the files before renaming the file type. By default, the POLYCENTER Software Installation utility does not preserve existing file versions (no archive). You cannot use this option with the release merge or write options.

assemble copy
Establishes the contents of the file by duplicating a file in the reference copy. This is the default.

assemble execute *(command,...)*
Establishes the contents of the file by executing the specified commands. Specify the command lines as a quoted or unquoted strings.

assemble uses *(file,...)*
Specifies a list of additional files required by the assemble execute option. Specify the relative file specification. By default, the assemble execute option does not require additional files.

assemble [no] requires file *(file,...)*
Specifies a list of files that you must assemble before the commands specified by the assemble execute option can execute. Specify the relative file specification. By default, the assemble execute option does not require additional files.

Note that required files are assembled before the file that requires them. For example, you can use this option to specify a main image that requires a shareable image.

[no] generation *generation*
Specifies that the file has an explicit generation number. Specify the number as an unsigned integer. Refer to the Description section for the meaning of this value. By default, the file does not have an explicit generation number (no generation).

image library
Specifies that the file's symbols are inserted into the system shareable image symbol table library. The file must be a shareable image.

image module *module_name*
Specifies the name of the module being inserted into the system shareable image symbol table library. If you do not specify this option, the module name is the same as the file name. This option has no effect unless specified with the image library option.

owner *owner*
Specifies the account name that owns the file. By default, the SYSTEM account owns the file.

protection execute
Sets the file protection, giving general users execute access.

protection private
Sets the file protection, giving general users no access.

protection public
Sets the file protection, giving general users read and execute access. This is the default.

release execute *(command,...)*
Specifies command lines (as quoted or unquoted strings) that execute to convert the file during a version upgrade. You cannot use this option with the release merge or release replace options.

release uses *(file,...)*
Specifies a list of additional files required by the release execute option. Specify the file specifications of the files. By default, the release execute option does not require additional files.

release merge
Specifies that library modules propagate during a version upgrade. If modules are present in the existing library but not in the new library, they are propagated to the new library. The file you specify with the **name** parameter must be a library. You cannot use this option with the archive, release execute, release replace, or write options.

release replace
Specifies that previous versions of the file are replaced during a version upgrade. You cannot use this option with the release execute or release merge options.

release notes
Specifies that the file is a release notes file. Users can extract the release notes to a file using the DCL command PRODUCT EXTRACT RELEASE_NOTES.

size *size*
Do not specify this option in your PDF. When you package your product, the utility calculates the size (in blocks) of the files you specify and provides this option in the output PDF.

source *source*
Specifies the relative file specification of the file in the reference copy as a single quoted or unquoted string. Use this option when you want to give a file a different name in the execution environment. When users install your product, the utility uses this source file in the reference copy to create the file you specify with the **name** parameter. By default, the file in the reference copy and the file created in the execution environment have the same file specification.

[no] write
Specifies that you expect that users will modify the file during system operation. If you specify this option, during a version upgrade if the file already exists it remains the active version. The default is no write. You cannot use this option with the archive or release merge options.

Description

The *file* statement creates a file in the execution environment. If a file of the same name already exists, the utility determines which file to preserve by comparing the generation numbers of the files.

If both files have generation numbers, the utility preserves the file with the higher generation number. If one file does not have a generation number or has a generation number of zero, the utility preserves the file that has a nonzero generation number.

The *file* statement specifies a file managed object. You specify the name of the file managed object using the **name** parameter. The name is a multicomponent name qualified by the relative directory path.

The *bootstrap block*, *link*, and *loadable image* statements can also specify references to a file managed object.

The file specified by the source option, if present, or otherwise by the **name** parameter, must be supplied as product material if the assemble copy option is in effect.

Each file specified by the assemble uses option must be supplied as product material if the assemble execute option is in effect.

The scope and lifetime of the file managed object depend on whether it is contained within a scope group (and the state of the assemble option) as shown in Table PDF–2.

Table PDF–2 File Managed Object Scope and Lifetime

Type of Scope Group	Lifetime	Scope
Product[1]	Product	Product
Global	Assembly	Global
Bootstrap	Operating	Bootstrap
Operating	Operating	Operating
Processor	Operating	Processor

[1]If the assemble option is not assemble copy, the file managed object has assembly lifetime and product scope.

If you specify the access control option, the *file* statement specifies one access control entry (ACE) managed object that references the file managed object for each entry specified with the access control option. The ACE managed object has the following characteristics:

- It is unnamed.
- It has operating lifetime. It has the same scope as the file managed object.
- The system resolves managed object conflict by managed object collection.

If the image library option is in effect, it specifies an image library module managed object that references the file managed object. The image library module managed object has the following characteristics:

- It must be unique within the global scope.
- It has assembly lifetime and global scope.
- Managed object conflict is not recoverable.

See Also

bootstrap block
directory
link
loadable image
scope

Examples

1. `file [SYSLIB]FDVSHR.EXE image library ;`

   The *file* statement in this example specifies that the symbols for the shareable image [SYSLIB]FDVSHR.EXE are inserted into the system shareable image symbol table library.

2. `file [SYSMGR]DECW$STARTUP.COM protection public ;`

   The *file* statement in this example creates the file [SYSMGR]DECW$STARTUP.COM, giving users read and execute access.

3. ```
   file [SYSMGR]DECW$SYLOGIN.COM protection public
       source [SYSMGR]DECW$SYLOGIN.TEMPLATE ;
   ```

   The *file* statement in this example creates the file [SYSMGR]DECW$SYLOGIN.COM in the execution environment using the contents of the file [SYSMGR]DECW$SYLOGIN.TEMPLATE from the reference copy.

4. `file [SYSMGR]DECW$SYSTARTUP.COM generation 56 archive write ;`

   The *file* statement in this example creates the file [SYSMGR]DECW$SYSTARTUP.COM, preserving the existing copy. It also assigns a generation number to the file for conflict resolution. For example, if a version of the file already exists with a generation number of 60, the utility will preserve the copy with generation number 60 and will not create a new one. The write option indicates that users can modify this file and that, during a version upgrade, the existing version should remain the active one.
   ```

hardware device

The *hardware device* statement specifies that a required hardware device must be present in the execution environment. If the device is not available, the POLYCENTER Software Installation utility prompts the user to continue or to terminate the operation.

Statement Syntax

hardware device *name* ;

Function Syntax

< hardware device *name* > ;

Parameter

name
Specifies the device name of the hardware device. You must include the colon (:) at the end of the device name.

Description

The *hardware device* statement specifies a required hardware device. If the device is not available, the utility prompts interactive users to continue or to terminate the operation. Each failed *hardware device* statement results in a separate prompt.

If the operation is not executed in interactive mode, it is terminated.

Function

The *hardware device* function tests whether a specified device is present. The value is true if the device is present. If the function value is true, a reference to the device is created. Otherwise, no reference is created.

The name of the referenced device is the value of the **name** parameter.

While the reference exists, the utility does not permit an operation that makes the specified conditions false.

Examples

1.
```
hardware device LPA0: ;
```

   The *hardware device* statement in this example specifies that if the device named LPA0: is not present in the execution environment, the utility displays a message prompting the user to continue or to terminate the operation.

2.
```
if (<hardware device GAA0:>) ;
    file [SYSEXE]SMFDRIVER.EXE ;
end if ;
```

   The *hardware device* function in this example provides the file [SYSEXE]SMFDRIVER.EXE if the device GAA0: is present.

hardware processor

The *hardware processor* statement specifies specific system processor models that must be present. If none of the specified system processor models is present, the POLYCENTER Software Installation utility prompts the user to continue or to terminate the operation.

Statement Syntax

hardware processor model *(model,...)* ;

Function Syntax

< hardware processor model *(model,...)* > ;

Parameter

model *(model,...)*
Specifies processor model identifiers.

Description

The *hardware processor* statement specifies specific system processor models. If none of the specified models is present, the utility prompts interactive users to continue or to terminate the operation. Each failed *hardware processor* statement results in a separate prompt.

If the operation is not executed in interactive mode, it is terminated.

Function

The *hardware processor* function tests whether a specified system processor model is present. The value is true if the model is present. If the function value is true, a reference to the system processor model is created. Otherwise, no reference is created.

The referenced system processor model is unnamed. While the reference exists, the utility does not permit an operation that makes the specified conditions false.

Example

Suppose the PDF contains the following lines:

```
if (<hardware processor model 7>) ;
    error UNSPROC ;
end if ;
```

You would have an UNSPROC module in the PTF similar to the following:

```
1 UNSPROC
=prompt Not supported on MicroVAX I.
This product is not supported on the MicroVAX I processor.
```

If the processor model is 7, the system displays a message indicating that the product is not supported on the MicroVAX I computer and prompts the user to continue or terminate the operation.

if

The *if* statement allows you to conditionally process a group of statements. If the expression evaluates to true, the POLYCENTER Software Installation utility processes only those statements lexically contained in the if group up to the first occurrence of an *else*, *else if*, or *end if* statement.

Syntax

if *expression* ; [else if *expression* ; | else ;]

Parameter

expression
Specifies the condition you want to test. This parameter must be an expression.

Required Terminator

end if ;

Description

The *if* statement allows you to conditionally process a group of statements. If the expression evaluates to true, the utility processes only those statements lexically contained in the if group up to the first occurrence of an *else* statement, *else if* statement (if present), or *end if* statement.

else if
The *else if* statement is valid only if it is immediately contained in an if group and it is not lexically preceded by an *else* statement.

The utility does not evaluate the expression in the *else if* statement if one of the following conditions exists:

- The result of evaluating the expression in the *if* statement is true.
- The result of evaluating the expression in any lexically preceding *else if* statement in the same if group is true.

If either of these conditions exists, the utility also does not execute statements lexically contained in the if group between the *else if* statement and the next occurrence of an *else*, *else if*, or *end if* statement.

The utility processes statements lexically contained in the if group between the *else if* statement and the next occurrence of an *else*, *else if*, or *end if* statement if the following conditions exist:

- The result of evaluating the expression in the *if* statement is false.
- The result of evaluating the expression in all lexically preceding *else if* statements in the same if group (if present) is false.
- The result of evaluating the *else if* expression is true.

else

The *else* statement is valid only if it is immediately contained in an if group and it is the only *else* statement in the if group. The utility executes the statements following the *else* statement (in the same if group) if both of the following conditions exist:

- The expression in the *if* statement evaluates to false.
- The expressions in all lexically preceding *else if* statements in the same if group (if present) are false.

Example

```
if (<software DEC VAXVMS DECWINDOWS>) ;
    file [SYSEXE]PRO$DW_SUPPORT.EXE ;
else if (<software DEC VAXVMS MOTIF>) ;
    file [SYSEXE]PRO$MOTIF_SUPPORT.EXE ;
else ;
    file [SYSEXE]PRO$CC_SUPPORT.EXE ;
end if ;
```

This example uses the *if* statement in conjunction with the *software* function to determine which file to provide, as follows:

1. If DECwindows is present, the utility provides the file [SYSEXE]PRO$DW_SUPPORT.EXE.
2. If DECwindows is not present and DECwindows Motif is present, the utility provides the file [SYSEXE]PRO$MOTIF_SUPPORT.EXE.
3. If neither DECwindows nor DECwindows Motif is present, the utility provides the file [SYSEXE]PRO$CC_SUPPORT.EXE.

infer

The *infer* statement tests the target system to determine if a product or product version is available.

Note

The *infer* statement is valid only in a transition PDF.

Syntax

infer available from install *file* ;

infer available from logical name *logical_name* ;

infer version from *file* ;

Parameters

file
Specifies the relative file specification of the file you want to test.

logical_name
Specifies the logical name you want to test.

Description

The *infer* statement tests the target system to determine if a product or product version is available. This statement is valid only in a transition PDF.

There are several types of *infer* statements:

- The *infer available* statement tests the target system to determine if the product named in the =product directive of the transition PDF is available. If no *infer available* statement is present, the product is inferred to be available.
  - The *infer available from install* statement tests whether the product is available only if the specified file is installed as a known image. The *scope* statement controls execution of this statement; the test executes in the specified scope.
  - The *infer available from logical name* statement tests whether the product is available only if the logical name you specify has a translation.
- The *infer version* statement tests the target system to determine the presence and active version of the product named in the =product directive of the transition PDF. The product is inferred to be present if the specified file is present on the system and absent otherwise. If the product is present, the active version is inferred to be the internal version number of the specified file. The *scope* statement controls execution of this statement; the test executes in the specified scope.

See Also

scope

Examples

1. `infer available from logical name DOC$ROOT ;`

   The *infer available* statement in this example determines if the product is available by checking to see if there is a translation for the logical name DOC$ROOT. The name of the product that the statement is testing for is contained in the =product directive in the transition PDF.

2. `infer version from [SYSEXE]FORTRAN.EXE`

   The *infer version* statement in this example determines the active version of the product by checking to see if the file [SYSEXE]FORTRAN.EXE is present.

information

The *information* statement displays text from a specified text module in the PTF either before or after the execution of a POLYCENTER Software Installation utility operation.

Syntax

information *name* [[no] confirm | phase { after | before }] ;

Parameter

name
Specifies the name of the associated PTF text module. The name you specify must be unique among all names in the same product description.

Options

[no] confirm
Displays the contents of the text module and prompts the user for a response. The user can continue or terminate the operation. The confirm option does not have any effect in batch mode. The default is no confirm.

phase after
Displays the contents of the text module after the POLYCENTER Software Installation utility operation finishes. This option cannot be used with the phase before option.

phase before
Displays the contents of the text module during the configuration phase. This option is the default and cannot be used with the phase after option.

Description

The *information* statement displays text from a specified text module in the PTF either before or after the execution of a POLYCENTER Software Installation utility operation.

The POLYCENTER Software Installation utility processes *information* statements in lexical order. The utility displays prompt text and help text (if the user requests it).

You must supply product text in the associated product text module. The module must contain a =prompt directive.

Information statements that specify the phase after option do not display text if they are lexically contained in an option group that is not selected.

If you have *information* statements that specify the phase before option and they are lexically contained in a group with configuration choices, they are processed in lexical order and may be nested.

The *information* statement declares a name; it is not a variable.

The confirm option to the *information* statement causes the utility to prompt the user to continue or terminate the operation.

Example

Suppose the product text file for DEC Rdb for OpenVMS software contains the following lines:

```
1 RELEASE_NOTES
=prompt Release notes for Rdb/VMS available.
The release notes for Rdb/VMS are available in the file
SYS$HELP:RDBVMSV4.RELEASE_NOTES.
1 STOP_RDB_VMS_MONITOR
=prompt The Rdb/VMS monitor must be stopped before installation
The Rdb/VMS monitor must be stopped before Rdb/VMS may be installed.
Perform the following operation:
$ @SYS$MANAGER:RMONSTOP
```

The product description file could contain the following information statements:

```
information RELEASE_NOTES phase after ;
information STOP_RDB_VMS_MONITOR phase before confirm ;
```

If the user requests help, the first *information* statement displays the following text after the operation finishes:

```
Release notes for Rdb/VMS available.
The release notes for Rdb/VMS are available in the file
SYS$HELP:RDBVMSV4.RELEASE_NOTES.
```

If the user requests help, the first *information* statement displays the prompt text after the operation finishes:

```
Release notes for Rdb/VMS available.
```

If the user requests help, the second *information* statement displays the following text for the user during the configuration phase:

```
The Rdb/VMS monitor must be stopped before installation
The Rdb/VMS monitor must be stopped before Rdb/VMS may be installed.
Perform the following operation:
$ @SYS$MANAGER:RMONSTOP
Do you want to continue [YES]?
```

If the user does not request help, the utility displays the prompt text during the configuration phase:

```
The Rdb/VMS monitor must be stopped before installation
Do you want to continue [YES]?
```

Regardless of whether the help display option is set, the confirm option in the second statement forces the user to respond to the prompt before continuing.

link

The *link* statement specifies a second directory entry for a file or directory.

Syntax

link *name* from *source* [type hard] ;

Parameters

name
Specifies the file specification of the second directory entry.

from *source*
Specifies the file specification of an existing directory entry for the file or directory. The parameter string must be a single quoted or unquoted string. The referenced file or directory must be defined by a directory or *file* statement in the same product description.

Options

type hard
Specifies that a hard link be created. This is the default.

Description

The *link* statement specifies a second directory entry for a file or directory.

The scope and lifetime of the link managed object depend on whether it is contained in a scope group, as shown in Table PDF–3.

Table PDF–3 Link Managed Object Scope and Lifetime

Type of Scope Group	Lifetime	Scope
Product	Product	Product
Global	Assembly	Global
Bootstrap	Operating	Bootstrap
Operating	Operating	Operating
Processor	Operating	Processor

If the *link* statement is not contained in a scope group or is contained in a scope product group, the link managed object has product lifetime and product scope.

Managed object conflict is unrecoverable.

See Also

directory
file
scope

Example

```
link [SYSEXE]FMS.EXE from [SYS$EXE]FMS.EXE ;
```

The statement in this example specifies that the file [SYSEXE]FMS.EXE is linked to the file [SYS$EXE]FMS.EXE.

loadable image

The *loadable image* statement places an image into the system loadable images table, SYS$LOADABLE_IMAGES:VMS$SYSTEM_IMAGES.DATA, and also into SYS$UPDATE:VMS$SYSTEM_IMAGES.IDX for compatibility with the System Management utility (SYSMAN).

Syntax

loadable image *image* product *product* [step { init | sysinit } | message *text* | severity { fatal | success | warning }] ;

Parameters

image
Specifies the file name of the system loadable image. The name you specify must be defined in the same product description and must have bootstrap scope and product or assembly lifetime.

product *product*
Specifies the product mnemonic (as a single quoted or unquoted string of 1 to 8 characters) that uniquely identifies the loadable image. For user-written images, this should typically contain the string _LOCAL_.

Options

step init
Specifies that the system load the image during the INIT step of the booting process.

step sysinit
Specifies that the system load the image during the SYSINIT step of the booting process. This is the default.

message *text*
Specifies the message you want displayed using the severity option. The message must be a single quoted or unquoted string. Case is significant. By default, the severity option displays the message "system image load failed."

severity fatal
Specifies that if an error occurs while the image is being loaded, the system display the message and bugcheck; if no error occurs, processing continues.

severity success
Specifies that the system continue processing and not display a message regardless of whether an error occurs while the image is being loaded.

severity warning
Specifies that if an error occurs while the image is being loaded, the system displays the message and continues; if no error occurs, the system continues and does not display the message. This is the default.

Description

The *loadable image* statement places an image into the system loadable images table, SYS$LOADABLE_IMAGES:VMS$SYSTEM_IMAGES.DATA, and also into SYS$UPDATE:VMS$SYSTEM_IMAGES.IDX for compatibility with the System Management utility (SYSMAN).

The *loadable image* statement specifies a loadable image module managed object that has the following characteristics:

- It must be unique within the global scope.
- It has assembly lifetime and global scope.
- Managed object conflict is not recoverable.

The *loadable image* statement also refers to a file managed object specified using the image parameter.

See Also

file

Example

```
loadable image DDIF$RMS_EXTENSION product _LOCAL_
    message "DDIF Extension not loaded"
    severity warning ;
```

The statement in this example places the user-written image DDIF$RMS_EXTENSION in the system loadable images table. If an error occurs while loading this image, the system displays the error message "DDIF Extension not loaded."

module

The *module* statement creates one or more modules in a command, help, macro, object, or text library.

Syntax

module *file* type *type* module *module_name* [[no] generation *generation* | [no] globals | library *library* | [no] selective search | size *size*] ;

Parameters

file
Specifies the file specification of the file that contains the modules.

type *type*
The library type. Table PDF–4 lists the keywords you can specify with this parameter.

Table PDF–4 Library Types for Module Statement

Keyword	Library Type	Default
Command	Command definition library	[SYSLIB]DCLTABLES.EXE
Help	Help library	[SYSHLP]HELPLIB.HLB
Macro	Macro library	[SYSLIB]STARLET.MLB
Object	Object library	[SYSLIB]STARLET.OLB
Text	Text library	[SYSLIB]STARLETSD.TLB

module *module_name*
Specifies the list of module names you are specifying.

Options

[no] generation *generation*
Specifies that the file has an explicit generation number. Specify the number as an unsigned integer. Refer to the Description section for the meaning of this value. By default, the file does not have an explicit generation number (no generation).

[no] globals
Specifies whether the global symbol names of the modules you are inserting into an object library are included in the global symbol table. You can use this option with object libraries only. By default, the global symbols of the module are inserted into the global symbol table.

library *library*
Specifies the file specification of the library. The file you specify must be a library of the type you specified with the **type** parameter.

[no] selective search
Specifies whether the input modules being inserted into the library are available for selective searches by the linker (by default they are not). For more information about selective searches, see the *OpenVMS Linker Utility Manual*.

size *size*
Specifies the size of the file as an unquoted string that specifies an unsigned integer. The utility supplies this option during a package operation; the utility ignores it if it is supplied as input to a package operation.

Description

The *module* statement creates one or more modules in a command, help, macro, object, or text library.

The name of a module managed object is specified using the module option.

The module managed object has assembly lifetime, and its scope is the same as the library.

If a module of the same name already exists, the utility determines which module to preserve using the generation numbers. If both modules have generation numbers, the utility preserves the module with the higher generation number. If one module does not have a generation number or has a generation number of zero, the utility preserves the module that has a nonzero generation number.

Examples

1.
```
module [SYSUPD]CDD.CLD type command module CDD ;
```

   The statement in this example creates the command module CDD in the default command library [SYSLIB]DCLTABLES.EXE using the file [SYSUPD]CDD.CLD.

2.
```
module [SYSUPD]HELP.HLP type help module FOO_HELP ;
```

   The statement in this example creates the help module FOO_HELP in the default help library [SYSHLP]HELPLIB.HLB using the file [SYSUPD]HELP.HLP.

3.
```
module [SYSUPD]SPI$CONNECT.MAR type macro
    library [SYSLIB]LIB.MLB module TEST ;
```

   The statement in this example creates the macro module TEST in the macro library [SYSLIB]LIB.MLB using the file [SYSUPD]SPI$CONNECT.MAR.

4.
```
module [SYSUPD]COBRTL.OBJ type object module TEST2;
```

   The statement in this example creates the object module TEST2 in the default object library [SYSLIB]STARLET.OLB using the file [SYSUPD]COBRTL.OBS.

5.
```
module [SYSUPD]PROTOTYPE_BOOK.TXT type text
    library [SYSLIB]LPS$FONT_METRICS.TLB module FONT;
```

The statement in this example creates the text module FONT in the text library [SYSLIB]LPS$FONT_METRICS.TLB using the file [SYSUPD]PROTOTYPE_BOOK.TXT.

network object

The *network object* statement uses a command procedure to create DECnet network objects.

Syntax

network object *name* with *(parameters,...)* ;

Parameters

name
Specifies the name of the network object. The network object name is passed to the command procedure as P1.

with *(parameters,...)*
Specifies the list of parameters that are passed to the command procedure that creates the network object. Each parameter must be a single unquoted or quoted string that specifies P2 through P8, in order. Refer to the Description section for the meaning of the parameters.

Description

The *network object* statement uses a command procedure (SYS$UPDATE:PCSI$CREATE_NETWORK_OBJECT.COM) to create network objects. The parameters that you pass to the command procedure that creates the network objects are:

- P1 specifies the name of the network object (using the **name** parameter).
- P2 specifies the object number (for DECnet Phase IV systems only).
- P3 specifies the user name associated with the object (for DECnet Phase IV systems only). If you specify a user name, it must already exist.
- P4 specifies parameters for DECnet Phase IV objects that use the Network Control Protocol (NCP).
- P5 specifies parameters for DECnet Phase V objects that use the Network Control Language (NCL).
- P6 through P8 are not used.

When you remove a product that created network objects, the POLYCENTER Software Installation utility uses a command procedure (SYS$UPDATE:PCSI$DELETE_NETWORK_OBJECT.COM) to delete network objects associated with your product.

The *network object* statement specifies a network object managed object that has the following characteristics:

- Its name is the value of the **name** parameter. The name must be unique with respect to all network object names in the processor scope.
- It has operating lifetime and processor scope.
- Managed object conflict is not recoverable.

Example

```
network object x$x0 with ("/priv=cmkrnl") ;
```

In this example, the *network object* statement creates a network object with the CMKRNL privilege.

option

The *option* statement is a utility directive that specifies a group of elements that the user can choose to make either all present or all not present. This statement allows you to present options to the user.

Statement Syntax

option *name* [default *value*] ;

Function Syntax

< option *name* [default *value*]> ;

Parameter

name
Specifies, as a quoted or unquoted string, the name of the associated PTF text module and configuration choices. The name you specify must be unique among all names in the same product description.

Option

default *value*
Specifies the default value for the variable. The value must be either 0 (false), 1 (true), yes, no, true, or false; the default is 1 (true).

If you specify an *option* statement with the default value 0, and it contains other *option* statements, any defaults in the contained *option* statements apply only when the top-level *option* statement is selected.

Required Terminator

end option ;

Description

The *option* statement specifies a group of elements that the user can choose to make either all present or all not present.

You can nest option groups. The POLYCENTER Software Installation utility processes nested options in lexical order. An option group containing *option* statements must be selected and processed before you can select any of the options it contains.

The prompt text is a string that describes the option. The utility displays this text to the user when prompting for a value. If the user sets the session's help display option, the utility displays the accompanying text, which should describe each option in as much detail as necessary.

The utility gets the default value for an option from either:

- The *option* statement in the PDF (if you use the default option).
- The product database (if you use the /PRODUCER=CURRENT qualifier to the PRODUCT command). This qualifier causes the utility to use the default values for the current version of the product.

- The preconfiguration file (PCF), if you created one.

You must supply text in the associated product text module. The module must contain a =prompt directive.

Function

The option function returns true if the user selects the option and false if the user does not select the option. The user can select options during the configuration phase. All option functions default to 1 (true).

See Also

part

Examples

1.
```
option NET ;
    file [SYSEXE]NETSERVER.COM ;
    file [SYSEXE]NETSERVER.EXE ;
    file [SYSHLP]NCPHELP.HLB ;
    option NET_A default 0 ;
        file [SYSEXE]FAL.COM ;
        file [SYSEXE]FAL.EXE ;
        end option ;
    option NET_B ;
        file [SYSEXE]REMACP.EXE ;
        file [SYSMGR]RTTLOAD.COM ;
        file [SYS$LDR]CTDRIVER.EXE ;
        file [SYS$LDR]RTTDRIVER.EXE ;
        end option ;
    end option ;
```

If the product description file contains the lines above, the product text file contains the corresponding text:

```
1 NET
=prompt network support
This option allows you to participate in a DECnet network.
1 NET_A
=prompt incoming remote file access
This option allows file access from other nodes in a DECnet network.
1 NET_B
=prompt incoming remote terminal access
This option allows users on other nodes in a DECnet network to log
in.
```

The user must select option NET before NET_A or NET_B are available for selection. Therefore, NET is processed before NET_A or NET_B.

2.
```
if (<option A>) ;
        file [SYSEXE]A.EXE ;
else ;
        file [SYSEXE]B.EXE ;
end if ;
```

The product text file contains the corresponding text:

```
1 A
=prompt the X capability
This feature provides the A capability, but you will not get the B
capability.
```

In this example, if the user selected the A option, the utility provides the file [SYSEXE]A.EXE. Otherwise, the utility provides the file [SYSEXE]B.EXE.

part

The *part* statement allows you to specify a message about a group of elements that you want to display to users.

Syntax

part *name* ;

Parameter

name
Specifies, as a quoted or unquoted string, the name of the associated PTF text module. The name you specify must be unique among all names in the same product description.

Required Terminator

end part ;

Description

The *part* statement allows you to specify a message about a group of elements that you want to display to users. For example, you can display a message about a group of software products that you are providing as part of a platform.

You can nest part groups. The POLYCENTER Software Installation utility processes nested parts in lexical order.

If the user requests help, the utility displays the accompanying text, which describes each part in detail. The prompt text is a string that describes the part. The utility displays this text to the user.

You must supply text in the associated product text module. The module must contain a =prompt directive.

See Also

option

Example

Suppose the product description file contains the following lines:

```
part DESKTOP ;
software DEC AXPVMS DECPRINT
    version required V4.0 component ;
software DEC AXPVMS DOCUMENT
    version required V2.0 component ;
software DEC AXPVMS LSE
    version required V3.0 component ;
end part;
```

The product text file contains the corresponding text:

```
1 DESKTOP
=prompt Desktop Publishing Tools
This product provides the following desktop publishing products:
DECprint Printing Services software, VAX DOCUMENT software, and the VAX
Language Sensitive Editor (LSE) software are the required products that
comprise this platform.
```

This example shows how to use the *part* statement to display a message about the required software products that the desktop platform provides.

patch image

The *patch image* statement updates an executable image.

Syntax

patch image *name* with *source* ;

Parameters

name
Specifies the relative file specification of the executable image you want to update.

with *source*
Specifies the file specification of the file containing the update commands. The file that contains the update commands should contain OpenVMS Image File Patch Utility (PATCH) commands.

Description

The *patch image* statement updates an executable image. Use this statement when it is inconvenient to provide a new image.

You must supply the file containing the update commands as part of the product material.

The *patch image* statement specifies a maintenance managed object that has the following characteristics:

- Its name is the same as the **name** parameter of the product group in which the statement is lexically contained; it is a multicomponent name qualified by the relative file specification of the file that is being updated. It must be unique with respect to all maintenance managed objects in all scopes.
- It has assembly lifetime, and its scope is the same as that of the file being updated.
- Managed object conflict is unrecoverable.

Example

```
patch image [SYS$LDR]SYS.EXE with [SYSUPD]VERSION_PATCH.PAT ;
```

This statement provides a file, [SYSUPD]VERSION_PATCH.PAT, to patch the image [SYS$LDR]SYS.EXE.

patch text

The *patch text* statement updates a text file.

Syntax

patch text *name* with *source* ;

Parameters

name
Specifies the relative file specification of the text file you want to update.

with *source*
Specifies the file specification of the file containing the update commands (as a single quoted or unquoted string). The file that contains the update commands should contain SUMSLP commands.

Description

The *patch text* statement updates a text file. Use this statement when it is inconvenient to provide a new file.

You must supply the file containing the update commands as part of the product material. You must also supply the file that you want to update, but this file is not propagated to the reference copy. The POLYCENTER Software Installation utility uses it to calculate the input and output checksum values.

The *patch text* statement specifies a maintenance managed object that has the following characteristics:

- Its name is the same as the **name** parameter of the product group in which the statement is lexically contained; it is a multicomponent name qualified by the relative file specification of the file that is being updated. It must be unique with respect to all maintenance managed objects in all scopes.
- It has assembly lifetime, and its scope is the same as that of the file being updated.
- Managed object conflict is unrecoverable.

Example

```
patch text [SYSUPD]VMSINSTAL.COM with [SYSUPD]VMSINSTAL.SLP ;
```

This statement provides a file, [SYSUPD]VMSINSTAL.SLP, to patch the text file [SYSUPD]VMSINSTAL.COM.

process parameter

The *process parameter* statement displays a message to users about process parameter requirements. Note that the POLYCENTER Software Installation utility does not adjust process parameters.

Syntax

process parameter *name* { consume *value* | require *value* | [maximum *value*] [minimum *value*] } ;

Parameter

name
Specifies the process parameter name. The name you specify must be valid on the system where the product executes.

Options

consume *value*
Specifies that the process parameter must be increased by the specified value. The value must be a single unquoted string that specifies an unsigned integer value. This option is valid if the data type of the value is signed integer or unsigned integer. Use this option when the product consumes a resource that is controlled by the process parameter.

maximum *value*
Specifies that the process parameter must have a value less than or equal to the specified value. The value must be a single unquoted string that specifies an integer value. This option is valid if the data type of the value is signed integer or unsigned integer.

minimum *value*
Specifies that the process parameter must have a value greater than or equal to the specified value. The value must be a single unquoted string that specifies an integer value. This option is valid if the data type of the value is signed integer or unsigned integer.

require *value*
Specifies that the process parameter must have the specified value. The value must be a single string that specifies a value of the parameter's type. This option is valid for any parameter data type.

Description

The *process parameter* statement displays a message to users after the installation about process parameter requirements. Note that the utility does not adjust process parameters.

Example

```
process parameter ASTLM minimum 6;
process parameter BYTLM require 32768;
process parameter PRCLM maximum 2;
process parameter FILLM maximum 40;
```

These statements display a message to users that a process that executes the product must have the following process parameters:

ASTLM greater than or equal to 6
BYTLM set to 32768
PRCLM less than or equal to 2
FILLM less than or equal to 40

process privilege

The *process privilege* statement displays a message to users about process privilege requirements. Note that the POLYCENTER Software Installation utility does not adjust process privileges.

Syntax

process privilege *(name[,...])* ;

Parameter

name
Specifies the process privilege names as a list. The privileges you specify must be valid on the system where the product executes.

Description

The *process privilege* statement displays a message to users after the installation about process privilege requirements. Note that the utility does not adjust process privileges.

Example

```
process privilege (group, oper, tmpmbx, sysnam) ;
```

The statement in this example displays a message to the user that processes using the product must have the GROUP, OPER, TMPMBX, and SYSNAM privileges.

product

The *product* group statement specifies the product identification and other descriptive information about the product.

Syntax

product producer abi name version type

Parameters

producer
Specifies the legal owner of the software product.

abi
Specifies the Application Binary Interface (ABI) on which the product executes. The parameter must be a single quoted or unquoted string.

name
Specifies the product name or in the case of a patch or mandatory update kit, the name of the kit. Specify the product name with an *apply to* statement for patch and mandatory update kits. The combination of product name, producer, and ABI must be unique.

version
Specifies the product version.

type
Specifies the description type as one of the following keywords:

- **full**. A complete description that specifies application software that can be made available on a system (in addition to the operating system) to produce fully functional software.
- **operating system**. A complete description that can be made available on a system to produce fully functional software. Exactly one product that specifies this type must be available on a system.
- **partial**. A partial description that can update an existing version of the same product on a system to produce functionally updated software. The active version number of the product changes.
- **patch**. A partial description that can update an existing version of the same product on a system to produce functionally updated software. The active version number of the product does not change.
- **platform**. A complete description of a software system containing multiple separate products.
- **transition**. A partial description that supplies information about product versions that predate conversion to the POLYCENTER Software Installation utility or to another technology that can interoperate with the utility.
- **mandatory update**. A partial description that can update an existing version of the same product on a system to produce functionally updated software. The active version number of the product does not change. A product description of the mandatory update description type must contain an *apply to* statement to identify the product to which the update applies.

Option

operating system
Specifies a transition kit for an operating system. This option is valid only for transition kit types.

Required Terminator

end product ;

Description

The *product* group statement is a utility directive that specifies the product identification and other descriptive information about the product. It does not specify a managed object.

Examples

1.
```
product DEC AXPVMS FMS V2.4 full ;
    file [sysmsg]fdvshr.exe image library ;
    file [sysmsg]fmsmsg.exe ;
    file [sysexe]fmsfed.exe ;
    file [sysexe]fmsfaa.exe ;
    file [sysexe]fmsfte.exe ;
    directory [systest.fms] ;
    file [systest.fms]ivp.exe ;
    file [systest.fms]samp.flb ;
end product ;
```

The *product* statement in this example specifies information about the FMS product.

2.
```
product DEC AXPVMS VMS V6.1 transition operating system ;
```

The *product* statement in this example specifies information about the OpenVMS AXP operating system. The operating system option identifies the software as an operating system transition kit.

register module

The *register module* statement registers in the product database one or more existing modules in a command, help, macro, object, or text library.

Syntax

register module type *type* module (*module_name*,...)

[[no] generation *generation*]
[library *library*] ;

Parameters

type *type*
Specifies the library type. Table PDF–5 lists the keywords you can specify with this parameter.

Table PDF–5 Library Types for Register Module Statement

Keyword	Library Type	Default
Command	Command definition library	[SYSLIB]DCLTABLES.EXE
Help	Help library	[SYSHLP]HELPLIB.HLB
Macro	Macro library	[SYSLIB]STARLET.MLB
Object	Object library	[SYSLIB]STARLET.OLB
Text	Text library	[SYSLIB]STARLETSD.TLB

module *module_name*
Specifies the names of the modules contained within the library. If you are registering a text module, you can specify only one module name.

Options

[no] generation *generation*
Specifies that the module has an explicit generation number. Specify the number as an unsigned integer. Refer to the Description section of the *module* statement for the meaning of this value. By default, the module does not have an explicit generation number (no generation).

library *library*
Specifies the file specification of the library. The file you specify must be a library of the type you specified with the **type** parameter.

Description

The *register module* statement registers in the product database one or more existing modules in a command, help, macro, object, or text library. Registering these modules in the product database allows the utility to detect conflicts with other modules.

Examples

1.
```
register module type help generation 5
        module (":=","=","@",ACCOUNTING,ALLOCATE,ANALYZE,APPEND,...) ;
```

   In this example, the *register module* statement registers several help modules. The generation option allows the utility to perform conflict resolution with these help modules.

2.
```
register module type object generation  1
       module (BAS$$CB,BAS$$COPY_FD,BAS$$DISPATCH_T,...) ;
```

   In this example, the *register module* statement registers several object modules. The generation option allows the utility to perform conflict resolution with these object modules.

rights identifier

The *rights identifier* statement uses a command procedure to create rights identifiers.

Syntax

rights identifier *name* with *(parameters,...)* ;

Parameters

name
Specifies the name of the rights identifier. The rights identifier name is passed to the command procedure as P1.

with *(parameters,...)*
Specifies the list of parameters that are passed to the command procedure that creates the rights identifier. Each parameter must be a single unquoted or quoted string that specifies P2 and P3, in order. Refer to the Description section for the meaning of the parameters.

Description

The *rights identifier* statement uses a command procedure (SYS$UPDATE:PCSI$CREATE_RIGHTS_IDENTIFIER.COM) to create rights identifiers. You pass the following parameters to the command procedure:

- P1 specifies the name of the rights identifier (using the **name** parameter).
- P2 specifies the optional qualifiers to use with the Authorize utility (AUTHORIZE) ADD/IDENTIFIER command.
- P3 specifies the the /VALUE qualifier to use with AUTHORIZE command ADD/IDENTIFIER. You can specify this parameter only if the identifier does not already exist on the system.

When you remove a product that created rights identifiers, the POLYCENTER Software Installation utility uses a command procedure (SYS$UPDATE:PCSI$DELETE_RIGHTS_IDENTIFIER.COM) to delete rights identifiers associated with your product.

The *rights identifier* statement specifies a rights identifier managed object that has the following characteristics:

- Its name is the value of the **name** parameter. The name must be unique with respect to all rights identifier names in the operating scope.
- It has operating lifetime and operating scope.
- Managed object conflict is not recoverable.

Example

```
rights identifier a$server with ("/priv=cmkrnl") ;
```

In this example, the *rights identifier* statement creates a rights identifier with the CMKRNL privilege.

remove

The *remove* statement removes managed objects from the product database and the system.

Note

You cannot use the *remove* statement in a transition PDF.

Syntax

remove ;

Required Terminator

end remove ;

Description

The *remove* statement allows you to remove managed objects from the product database and from the system. Statements that normally provide managed objects (for example, the *file* statement) remove managed objects when contained within a remove group.

By using the *remove* statement in a partial, patch, or mandatory update kit, you can eliminate obsolete files from a previous version of your product. By using the *remove* statement in a full kit, you can eliminate objects provided by a previous installation mechanism (for example, VMSINSTAL).

Statements that do not provide managed objects function normally within a remove group.

You can nest remove groups with scope groups, if necessary.

Examples

1.
```
remove ;
  directory [SYSHLP.EXAMPLES.FOO] ;
  file [SYSHLP.EXAMPLES.FOO]SMLUS.COM ;
  file [SYSHLP.EXAMPLES.FOO]SMLUT.COM ;
  file [SYSHLP.EXAMPLES.FOO]SMLUU.COM ;
end remove ;
```

The statements in this example remove some files and a directory (if they exist) from the product database and the running system.

2.
```
scope bootstrap ;
  remove ;
  file [SYSEXE]PROD_PROC.EXE ;
  end remove ;
  file [SYSEXE]PROD_PROC_V2.EXE ;
end scope ;
```

The statements in this example remove a file in the bootstrap scope and then provide a new file.

scope

The *scope* statement specifies for certain managed objects a scope and lifetime other than its defaults.

Syntax

scope bootstrap ;

scope global ;

scope processor ;

scope product ;

Required Terminator

end scope ;

Description

The *scope* statement specifies for a managed object a scope and lifetime other than its defaults. You can nest *scope* statements. For more information about the scope and lifetime of managed objects, see Appendix B.

See Also

directory
execute install, remove
execute release
file
infer
link

Example

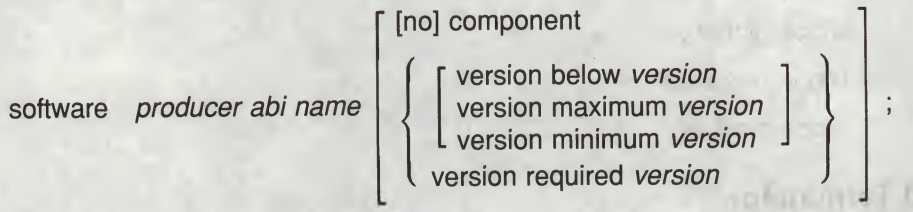
```
scope bootstrap ;
    file [SYSEXE]SYSBOOT.EXE ;
    file [SYSEXE]VMB.EXE ;
    bootstrap block [SYSEXE]VMB.EXE image [SYSEXE]BOOTBLOCK.EXE ;
end scope;
```

The statements in this example specify that the files VMB.EXE and SYSBOOT.EXE must be placed on every bootstrap disk.

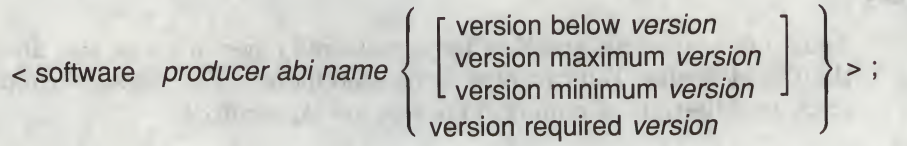
software

The *software* statement specifies a software product that must be available. The *software* function tests for the presence of a second product. You can also specify the version of the product that must be present.

Statement Syntax



Function Syntax



Parameters

producer
Specifies the legal owner of the software product.

abi
Specifies the Application Binary Interface (ABI) on which the product executes. The parameter string must be a single quoted or unquoted string.

name
Specifies the product name. The combination of producer name, ABI, and product name must be unique.

Options

[no] component
Specifies that if the product is copied (using a copy operation) or packaged (using a package operation), the component products will be copied or packaged along with the product. The default is no component (the product does not need to be present during a copy or package operation).

version below *version*
Specifies the smallest invalid product version. Use this option to specify that the product version must be less than (but not equal to) the specified version. You cannot use this option with either the version minimum or version required options. By default, there is no smallest invalid version.

version maximum *version*
Specifies a maximum product version. The version must be a single quoted or unquoted string that specifies a version identifier. The version of the product that is available must be less than or equal to the specified version. You cannot

use this option with the version below option. By default, there is no maximum version.

version minimum *version*
Specifies a minimum product version. The version must be a single quoted or unquoted string that specifies a version identifier. The version of the product that is available must be greater than or equal to the specified version. By default, there is no minimum version.

version required *version*
Specifies a required product version. The version must be a single quoted or unquoted string that specifies a version identifier. The version of the product that is available must be equal to the specified version. You cannot use this option with either the version below, version maximum, or version minimum options. By default, there is no required version.

Description

The *software* statement specifies a software product that must be available in the execution environment. You can also specify a specific version of a product.

If you use the component option, the POLYCENTER Software Installation utility creates a copy of the referenced product when you copy your product.

If you use the version minimum or version maximum option, the POLYCENTER Software Installation utility searches in the following order the first character of the version identifier of available products for a match:

V
T
E
X
S
D
B
null

If the utility finds a match, it compares the rest of the version identifier to the constraints you specified with the version minimum or version maximum option.

If you reference a product that is not available or one that does not fit the constraints you specified, the utility prompts the user to continue or to terminate the operation.

If the operation executes in batch mode and a referenced product is not available, the operation terminates.

Note

If a referenced product is not available, Digital recommends that users accept the default prompt and terminate the operation.

If your product references another product with a *software* statement, the referenced product will be installed earlier than, and removed later than, your product. If two products reference each other (creating an infinite loop), the utility issues an error message.

Function

The *software* function tests for the presence of a second product. You can also specify a specific version of a product. The software function tests the state the system will be in when the operation finishes, not when the operation begins.

The function value is true if the following conditions exist; otherwise, the value is false:

- The product specified by the **producer**, **abi**, and **name** parameters is available.
- The version option is omitted, or the available version satisfies the specified constraints.

If the function value is true, the utility creates a reference to the product. While the reference exists, the utility does not permit an operation that makes the specified conditions false. If the function value is false, the utility does not create a reference.

Examples

1.
```
software DEC VAXVMS FORTRAN
    version minimum V3.0 version maximum V5.0 ;
```

   The *software* statement in this example specifies that this product requires DEC Fortran software. The version must be between 3.0 and 5.0.

2.
```
software DEC VAXVMS FORTRAN
    version below V5.0 ;
```

   The *software* statement in this example specifies that this product requires DEC Fortran software. The version must be less than (but not equal to) 5.0.

system parameter

The *system parameter* statement allows you to display a message to users that expresses system parameter requirements for your product. Note that the POLYCENTER Software Installation utility does not change system parameters.

Syntax

system parameter *name* { consume *value* | [maximum *value* | minimum *value*] | require *value* } ;

Parameter

name
Specifies the name of the system parameter. The parameter you specify must be valid on the system where the product executes.

Options

consume *value*
Specifies that the value of the system parameter must be increased by the specified value. Use this option when the product consumes a resource that is controlled by the system parameter. The value must be a single unquoted string that specifies an unsigned integer value. This option is valid if the data type of the value is signed integer or unsigned integer. If you specify this value, you cannot specify the minimum, maximum, or require option.

maximum *value*
Specifies that the system parameter must have a value less than or equal to the specified value. The value must be a single unquoted string that specifies an integer value. This option is valid if the data type of the value is signed integer or unsigned integer.

minimum *value*
Specifies that the system parameter must have a value greater than or equal to the specified value. The value must be a single unquoted string that specifies an integer value. This option is valid if the data type of the value is signed integer or unsigned integer.

require *value*
Specifies that the system parameter must have the specified value. The value must be a single string that specifies a value of the parameter's type.

Description

The *system parameter* statement displays a message to users after the installation about system parameter requirements for your product. Note that the POLYCENTER Software Installation utility does not adjust system parameters.

Example

```
system parameter vaxcluster require 1 ;
system parameter tty_classname require "TT" ;
system parameter pagedyn consume 200 ;
```

The statements in this example display the following messages:

```
This product requires the following system parameters
parameter VAXCLUSTER require 1

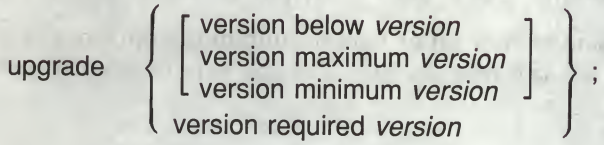
This product requires the following system parameters
parameter TTY_CLASSNAME require TT

This product requires the following system parameters
parameter PAGEDYN consume 200
```

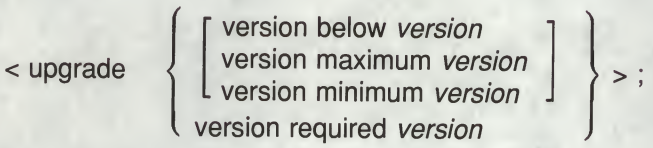
upgrade

The *upgrade* statement specifies product versions that must be available for a successful product upgrade. The *upgrade* function tests whether a version of the product is available.

Syntax



Function Syntax



Options

version below *version*
Specifies the smallest invalid product version. Use this option to specify that the product version must be less than (but not equal to) the specified version. You cannot use this option with either the version minimum or version required options. By default, there is no smallest invalid version.

version maximum *version*
Specifies a maximum product version that must be available. Use this option to specify that the product version must be less than or equal to the version you specify. You cannot use this option with the version below option. By default, there is no maximum version.

version minimum *version*
Specifies a minimum product version that must be available. Use this option to specify that the product version must be greater than or equal to the version you specify. By default, there is no minimum version.

version required *version*
Specifies a required product version that must be available. Use this option to specify that a specific product version must be present. You cannot use this option with either the version below, version maximum, or version minimum options. By default, there is no required version.

Description

In a full, operating system, or platform PDF, the *upgrade* statement specifies which product versions must be available for a successful product upgrade. If there is no available version of the product, the POLYCENTER Software Installation utility ignores the *upgrade* statement and performs a new installation.

In a partial PDF, the *upgrade* statement is required to specify which product versions must be available in order for the partial kit to be applied successfully.

If you use the version minimum or version maximum option, the utility searches in the following order the first character of the version identifier of available products for a match:

V
T
E
X
S
D
B
null

If the utility finds a match, it compares the rest of the version identifier to the constraints you specified with the version minimum or version maximum option.

If the *upgrade* statement is not present, there is no constraint on the available version of the product.

Function

The *upgrade* function tests whether a version of the product is available. If a version of the product is available, the function returns True. If a version of the product is not available, the function returns False.

See Also

apply to
product
software

Examples

1.
```
product DEC VAXVMS VMS V5.5-2 partial ;
upgrade version minimum V5.5 version maximum V5.5-2 ;
   .
   .
   .
 file "[SYSUPD]DEC-VAX-VAX-V0505--2.SPIU$EXECUTE_INSTALL" ;
 file "[SYSUPD]DEC-VAX-VAX-V0505--2.SPIU$EXECUTE_REMOVE" ;
   .
   .
   .
end product;
```

The *upgrade* statement in this example specifies that to install VMS Version 5.5–2, your system must be running at least VMS Version 5.5 and no higher than VMS Version 5.5–2.

2.
```
product DEC AXPVMS VMS V2.0 operating system ;
 upgrade version minimum V1.0 version maximum V2.0 ;
   .
   .
   .
 file "[SYSUPD]DEC-AXP-AXP-V0505--2.SPIU$EXECUTE_INSTALL" ;
 file "[SYSUPD]DEC-AXP-AXP-V0505--2.SPIU$EXECUTE_REMOVE" ;
   .
   .
   .
end product;
```

The *upgrade* statement in this example specifies that to install OpenVMS AXP Version 1.0, your system must be running at least OpenVMS AXP Version 1.0 and no higher than OpenVMS AXP Version 2.0.

A

Migrating from VMSINSTAL to the POLYCENTER Software Installation Utility

VMSINSTAL is an installation mechanism supplied by Digital. This appendix contains information about VMSINSTAL options and callbacks and their POLYCENTER Software Installation utility equivalents.

A.1 VMSINSTSAL Options and Equivalents

Table A–1 lists some tasks that you may need to perform, the corresponding VMSINSTAL option, and the POLYCENTER Software Installation utility equivalent. Note that some VMSINSTAL options do not have an equivalent. In many cases, this is because the design of the POLYCENTER Software Installation utility eliminates the need for an equivalent.

Table A–1 VMSINSTAL Options and Equivalents

Task	VMSINSTAL Option	POLYCENTER Software Installation Utility Equivalent
Creating a file that specifies answers to installation questions	OPTIONS A	Create a preconfiguration file (PCF) (see the *POLYCENTER Software Installation Utility User's Guide* manual). This is similar to an auto-answer file in VMSINSTAL.
Specifying a temporary work directory	OPTIONS AWD	Specify the /WORK qualifier to the PRODUCT command.
Startup	OPTIONS B[1]	No equivalent.
Tracing callbacks during installation	OPTIONS C[2]	Use the /LOG and /TRACE qualifiers to the PRODUCT command. You can also use the /NOCOPY qualifier when debugging a product description file (PDF) to prevent the product material from being copied into the reference copy.
Manipulating product kits	OPTIONS G	Use the COPY/FORMAT=REFERENCE and COPY/FORMAT=SEQUENTIAL commands to manipulate product kits (see Chapter 4).
Suppressing VMSINSTAL prompts	OPTIONS I[2]	No equivalent.
Debugging a kit	OPTIONS K[2]	Use the /LOG and /TRACE qualifiers to assist in debugging a PDF.
Providing a log of installation operations	OPTIONS L	Use the /LOG and /TRACE qualifiers. This provides more information than OPTIONS L with VMSINSTAL.

[1]OpenVMS startup use only

[2]Developer's use only

(continued on next page)

Table A–1 (Cont.) VMSINSTAL Options and Equivalents

Task	VMSINSTAL Option	POLYCENTER Software Installation Utility Equivalent
Displaying or printing release notes	OPTIONS N	Use the release notes option to the *file* statement and the PRODUCT EXTRACT RELEASE_NOTES command.
Performing an installation in test mode	OPTIONS Q[2]	No equivalent.
Installing a product in an alternate root	OPTIONS R	Use the /DESTINATION qualifier.
Pausing the installation at various points	OPTIONS RSP[2]	No equivalent.
Compiling information about the installation	OPTIONS S[2]	Use the /LOG and /TRACE qualifiers to the PRODUCT command.

[2]Developer's use only

A.2 VMSINSTAL Callbacks and Equivalents

To install a product using VMSINSTAL, you create a command procedure named KITINSTAL.COM that makes callbacks to VMSINSTAL. If you are migrating from VMSINSTAL to the POLYCENTER Software Installation utility, refer to Table A–2, which lists the VMSINSTAL callbacks and their equivalents.

Table A–2 VMSINSTAL Callbacks and Equivalents

Task	VMSINSTAL Callback	Option	POLYCENTER Software Installation Utility Equivalent
Adding an identifier to the rights database	ADD_IDENTIFIER		Use the *rights identifier* statement.
Prompting the installer for information	ASK		To confirm the completion of preinstallation tasks, use the confirm option to the *information* statement. The product text file (PTF) contains the prompt and help text.
Not recording responses to installation questions		A	No equivalent.
Forcing a Boolean answer		B	No equivalent.
Preceding prompt with blank line		D	No equivalent.
Disabling terminal echo		E	No equivalent.
Displaying help text before the prompt		H	The *information* statement.
Answer must be an integer		I	No equivalent.
Returning input in lowercase		L	No equivalent.

(continued on next page)

Table A–2 (Cont.) VMSINSTAL Callbacks and Equivalents

Task	VMSINSTAL Callback	Option	POLYCENTER Software Installation Utility Equivalent
Returning input in the same case		M	No equivalent.
Indicating a null response is acceptable		N	No equivalent.
Ringing the terminal bell before the prompt		R	No equivalent.
Indicating the response can be a string		S	No equivalent.
Returning input in uppercase		U	No equivalent.
Indicating the response can be Ctrl/Z		A	No equivalent.
Determining whether a license for the product is installed on the system	CHECK_LICENSE		No equivalent. License management is outside the domain of the utility.
Determining whether the network is running	CHECK_NETWORK		No equivalent. If you use a statement that references the DECnet network, the utility ensures that the network is available.
Determining whether there is sufficient disk space on the target device	CHECK_NET_ UTILIZATION		No equivalent. The utility ensures that sufficient disk space is available.
Determining whether a minimum version of software is present in the execution environment	CHECK_PRODUCT_ VERSION		Use the version minimum option to the *software* statement.
Limiting an installation to specified versions of the OpenVMS operating system	CHECK_VMS_ VERSION		Use the version minimum and version maximum options to the *software* statement, specifying DEC as the producer name, VAXVMS or AXPVMS as the abi, and VMS as the product name.
Determining which is the most recent version of an image	COMPARE_IMAGE		You can manage file versions using the generation option to the *file* statement.
Determining whether the user has loaded the license for the product being installed on the system	CONFIRM_LICENSE		No equivalent. License management is outside the domain of the utility.
Providing for orderly exit from an installation	CONTROL_Y		No equivalent necessary; the utility provides this automatically.
Creating an account on the system	CREATE_ACCOUNT		Use the *account* statement.

(continued on next page)

Table A–2 (Cont.) VMSINSTAL Callbacks and Equivalents

Task	VMSINSTAL Callback	Option	POLYCENTER Software Installation Utility Equivalent
Deleting obsolete files from a previous installation	DELETE_FILE		In full and operating system kits, the utility deletes files that are replaced during an upgrade. However, in a partial kit, you can remove obsolete files using the *remove* statement.
Locating files	FIND_FILE		If you want to determine whether an optional software product is available, use the *software* function. You do not need to determine whether a file is present before performing an operation that references it; the utility does this automatically.
Generating Structure Definition Language (SDL) definition files	GENERATE_SDL		No equivalent.
Extracting the image file identification string for a file	GET_IMAGE_ID		If you want to determine the available version of a software product, use the *software* statement or function.
Obtaining a password for an account	GET_PASSWORD		No equivalent necessary; the utility provides this function.
Placing requirements on system parameters	GET_SYSTEM_PARAMETER		Use the *system parameter* statement.
Displaying messages to the user	MESSAGE		Use the *information* statement to display information about pre- and postinstallation tasks. You do not need to provide error messages and progress information; the utility does this automatically.
Patching an image as part of the installation	PATCH_IMAGE		Use the *patch image* statement.
Moving a shareable image's symbol table to the system shareable image library when the patch is complete		I	No equivalent necessary. The image library option to the *file* statement controls its replacement in the image library.
Creating a journal file of patches		J	No equivalent.

(continued on next page)

Table A–2 (Cont.) VMSINSTAL Callbacks and Equivalents

Task	VMSINSTAL Callback	Option	POLYCENTER Software Installation Utility Equivalent
Saving old versions of the image file		K	No equivalent necessary. The utility deletes existing versions.
Moving the file to the SYS$SPECIFIC directory		O	No equivalent necessary. The placement of the *file* statement that originally described the image within a scope group determines its placement.
Reinstalling the image when the patch is complete		R	No equivalent necessary; the utility does this automatically.
Queuing a print job to SYS$PRINT	PRINT_FILE		No equivalent.
Invoking a command procedure of product-specific callbacks	PRODUCT		No equivalent.
Adding a command to the system DCL table	PROVIDE_DCL_COMMAND		Use the *module* statement with the **type command** parameter. You do not need to reinstall the system command table as a known image; the utility does this automatically.
Adding help to the DCL help library	PROVIDE_DCL_HELP		Use the *module* statement with the **type help** parameter.
Adding a new file to the system	PROVIDE_FILE		Use the *file* statement.
Placing the file in more than one location		C	No equivalent necessary.
Preserving old versions		K	No equivalent necessary. The utility deletes existing versions.
Adding the file to the SYS$SPECIFIC directory		O	Enclose the *file* statement in a scope processor group.
Specifying an input file that contains a list of logical names for the source files and their respective destinations		T	No equivalent necessary. Use one *file* statement for each file.
Adding a new image to the system	PROVIDE_IMAGE		Use the *file* statement. The utility can distinguish whether a file is a valid executable image.
Placing the file in more than one location		C	No equivalent necessary.

(continued on next page)

Table A–2 (Cont.) VMSINSTAL Callbacks and Equivalents

Task	VMSINSTAL Callback	Option	POLYCENTER Software Installation Utility Equivalent
Dynamically patching ECOs into the new image file		E	No equivalent necessary. You should package the file with the correct ECO numbers already set.
Moving a shareable image's symbol table to the system shareable image library		I	Use the image library option to the *file* statement.
Preserving old versions		K	No equivalent necessary. The utility deletes existing versions.
Moving the file to the SYS$SPECIFIC directory		O	Enclose the *file* statement in a *scope processor* group.
Specifying an input file that contains a list of logical names for the source image files and their respective destinations		T	No equivalent necessary. Use one *file* statement for each file.
Changing the file name and file type of all versions of a file	RENAME_FILE		Use the archive option of the *file* statement to preserve an existing version of a file during an upgrade.
Restoring save sets of a product that is divided among several save sets	RESTORE_SAVESET		No equivalent necessary.
Running an image during installation	RUN_IMAGE		Use the *execute* statement or the assemble execute or release execute option to the *file* statement.
Specifying a UIC or protection code for product files	SECURE_FILE		Use the owner and protection options to the *directory* and *file* statements.
Modifying the access control list (ACL) of a device, directory, or file	SET	ACL	Use the access control option of the *file*, *hardware device* and *directory* statements.
Determining the default case (upper or lower) in which text from the installer is returned to the installation procedure	SET	ASK_CASE	No equivalent.
Running an installation verification procedure (IVP)	SET	IVP	No equivalent necessary. You can specify the *execute test* statement and invoke the functional test for a product with the /TEST qualifier to the PRODUCT command.

(continued on next page)

Table A–2 (Cont.) VMSINSTAL Callbacks and Equivalents

Task	VMSINSTAL Callback	Option	POLYCENTER Software Installation Utility Equivalent
Calling a product's installation procedure after files have been moved to their target directories	SET	POSTINSTALL	Depending on your application, you can use the *execute postinstall* statement.
Purging files replaced by an installation	SET	PURGE	No equivalent necessary. The utility deletes existing versions.
Rebooting the system after the installation	SET	REBOOT	No equivalent.
Ensuring a high level of installation success	SET	SAFETY	No equivalent necessary. The utility provides the necessary disk management and reliability features.
Rebooting the system after the installation	SET	SHUTDOWN	No equivalent.
Specifying a product-specific startup command procedure	SET	STARTUP	Use the *execute start* statement.
Editing text files	SUMSLP_TEXT		Use the *patch text* statement.
Identifying installation peculiarities	TELL_QA		No equivalent necessary.
Exiting the installation procedure	UNWIND		No equivalent necessary. The utility controls the flow of the installation.
Updating an existing user account	UPDATE_ACCOUNT		Use the *account* statement to modify existing user accounts.
Making a file available for updating by copying it to a working directory	UPDATE_FILE		No equivalent necessary.
Modifying an identifier in the rights database	UPDATE_IDENTIFIER		Use the *rights identifier* statement to modify an existing rights identifier.
Updating a library	UPDATE_LIBRARY		Use the *module* statement with the appropriate parameter for the type of library you are updating. To update the shareable image library, use the image library option to the *file* statement. No equivalent exists to update RSX libraries.

B

Advanced PDF Concepts

This appendix contains information about some advanced PDF concepts such as managed object scope and lifetime.

B.1 Defining the Scope of a Managed Object

The **scope** of a managed object defines the degree of sharing that the managed object permits. For example, some objects are available only to certain processes, and some can be shared by all processes. The POLYCENTER Software Installation utility usually ensures that managed objects have the correct scope.

You might need to give a managed object a scope other than its default. Using the *scope* statement, you can ensure that the managed object is placed in the correct area on the system and that processes that need to can access it.

This section describes the different scopes that managed objects have:

- **Global scope** is the largest scope in which a single POLYCENTER Software Installation utility operation can have an effect. A single file that must be shared by every process in the computing facility must exist in global scope. Modules in system object libraries are examples of managed objects that must be in global scope. Writable databases might be in global scope.

  Global scope managed objects have the following characteristics:

  - They exist in every process in the computing facility (as established by an **initial system load**).
  - They are always stored together.
  - They are usually used by operating system products.

- **Bootstrap scope** managed objects function during system bootstrap when operating system facilities are unable to locate and use larger scopes. Only an operating system product can define bootstrap scope managed objects. Drivers and loadable images that must be present before startup executes are examples of files that should be in the bootstrap scope.

  Bootstrap scope managed objects have the following characteristics:

  - They support specific computers and usually exist on the same disk volume as processor scope managed objects.
  - A computing facility can have several bootstrap scopes, and multiple computers can share the same bootstrap scope.
  - They are usually used by operating system products.

- **Product scope** managed objects are product specific. Most managed objects for a product reside in product scope. Product scope managed objects for different products can be stored together or separately.

- **Operating scope** managed objects exist in all processes in an **operating domain**. An operating domain is a subdivision of the computing facility. One operating scope exists in each operating domain.

  For example, you might have two user authorization files in a computing environment, each within its own operating scope.

  Operating scope managed objects have the following characteristics:

  - Operating scope managed objects for different products must be stored together.
  - Operating scope and global scope managed objects often have identical boundaries; they are also often established at different times. When you assemble an integrated platform, you can establish global scope managed objects centrally, but operating scope managed objects must be established at the point of use.
  - They are usually used by operating system products.

- **Processor scope** managed objects exist in all processes executing on a single computer. For example, a logical name might exist in processor scope.

B.2 Updating Files

When you update your product with a partial, patch, or mandatory update kit, you can either explicitly state the scope of the file managed objects you are updating or let the utility determine the scope of the file managed objects:

- You can use the *scope* statement to ensure that the utility looks in a specific scope for the file managed object you want to update.
- If you do not use the *scope* statement, the utility searches the execution environment for a file managed object with the same name. If the utility finds the object, it replaces the object; if the utility does not find the file managed object, it provides a new file in product scope.

If you use the *patch* statement, the object you are updating must have been provided by your product. If you use the *module* statement, the object you are updating either must have been provided by your product or must be in global or bootstrap scope.

B.3 Managed Object Lifetimes

The **lifetime** of a managed object defines the duration or time period in which the managed object exists. For example, some managed objects are created each time a product is started; others are created before the product starts.

This section lists the lifetimes that a managed object receives as a result of its scope:

- **Product lifetime** managed objects are part of the product material. You create them when you package the product, and they exist in all reference and sequential copies. The POLYCENTER Software Installation utility does not create these managed objects, but it may copy or delete them. For example, an executable image is usually a product lifetime managed object.
- **Assembly lifetime** managed objects are created before the product is started. They do not need to be created each time the product is started and are identical in every execution environment. For example, a module in a system object library is an assembly lifetime managed object.

- **Operating lifetime** managed objects are created in the execution environment, before the product is started. They do not need to be created each time the product is started. For example, a user account is an operating lifetime managed object.
- **Dynamic lifetime** managed objects are created each time the product is started. For example, a server process that runs continuously to handle inbound network connections is a dynamic lifetime managed object.

Glossary

This glossary lists and defines the terms used in this manual.

Application Binary Interface (ABI)

A designation that specifies the hardware and software combination that a product requires.

integrated platform

A strategic combination of software products that is targeted toward a specific market or a set of applications that a company has standardized for internal use.

managed object

An entity that exists to support the proper functioning of a product. Files, directories, and accounts are all examples of types of managed objects.

package operation

A POLYCENTER Software Installation utility operation that uses the PDF, PTF, and product material to create a reference or sequential copy.

patch

A minor update to a software product that does not change the version level of the product.

PCF

See *preconfiguration file*.

PDB

See *product database*.

PDF

See *platform product description file*, *product description file*.

PDL

See *product description language*.

platform product description file (PDF)

A text file that specifies the execution environment for an integrated platform. See also *integrated platform*.

platform product text file (PTF)

A text file that contains all the product-specific text that the POLYCENTER Software Installation utility can display during the manipulation of a platform (for example, installation text). See also *integrated platform*.

POLYCENTER Software Installation utility

A software product that allows you to create software kits and manage software (for example, installation, removal, configuration).

preconfiguration file (PCF)

A text file that specifies configuration choices for the POLYCENTER Software Installation utility to use in subsequent operations. For example, you can use a PCF to avoid specifying the same answers to installation questions when you have multiple installations to perform.

product database (PDB)

The repository in which the POLYCENTER Software Installation utility records information about events such as product installation and removal. Users can query the PDB to find out information about their environment.

product description file (PDF)

A text file that specifies the execution environment for your product.

product description language (PDL)

The set of statements that you use to write a PDF. See also *product description file*.

product material

The files associated with the product.

product text file (PTF)

A text file that contains all the product-specific text that the POLYCENTER Software Installation utility can display during product manipulation (for example, installation text).

PTF

See *platform product text file*, *product text file*.

reference copy

A ready-to-install version of your product (that the POLYCENTER Software Installation utility can access) that you create using a POLYCENTER Software Installation utility package or copy operation.

removal

An operation opposite to installation and that reverses the effect of an installation.

sequential copy

This an optional form your product takes as a result of a package or copy operation. It consists of a single container file that can be placed on a sequential-access device (for example, a magnetic tape drive).

transition product description file (PDF)

A type of PDF that allows you to reference products not converted to the POLYCENTER Software Installation utility and to migrate products to the POLYCENTER Software Installation utility.

update

A minor revision to a software product that changes the version level of the product.

utility directive

PDL statements that do not specify managed objects. Utility directives affect the operation of the POLYCENTER Software Installation utility but do not affect the execution environment.

Index

R

S

T

U

V

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-DIGITAL (800-344-4825) and press 2 for technical assistance.

Electronic Orders

If you wish to place an order through your account at the Electronic Store, dial 800-234-1998, using a modem set to 2400- or 9600-baud. You must be using a VT terminal or terminal emulator set at 8 bits, no parity. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825) and ask for an Electronic Store specialist.

Telephone and Direct Mail Orders

From	Call	Write
U.S.A.	DECdirect Phone: 800-DIGITAL (800-344-4825) Fax: (603) 884-5597	Digital Equipment Corporation P.O. Box CS2008 Nashua, NH 03061
Puerto Rico	Phone: (809) 781-0505 Fax: (809) 749-8377	Digital Equipment Caribbean, Inc. 3 Digital Plaza, 1st Street Suite 200 Metro Office Park San Juan, Puerto Rico 00920
Canada	Phone: 800-267-6215 Fax: (613) 592-1946	Digital Equipment of Canada Ltd. 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6 Attn: DECdirect Sales
International	———	Local Digital subsidiary or approved distributor
Internal Orders[1] (for software documentation)	DTN: 264-3030 (603) 884-3030 Fax: (603) 884-3960	U.S. Software Supply Business Digital Equipment Corporation 10 Cotton Road Nashua, NH 03063-1260
Internal Orders (for hardware documentation)	DTN: 264-3030 (603) 884-3030 Fax: (603) 884-3960	U.S. Software Supply Business Digital Equipment Corporation 10 Cotton Road Nashua, NH 03063-1260

[1]Call to request an Internal Software Order Form (EN–01740–07).

Reader's Comments

POLYCENTER Software Installation
Utility Developer's Guide
AA–Q28MA–TK

Your comments and suggestions help us improve the quality of our publications.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (product works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

_____

What I like best about this manual is _____

_____

_____

What I like least about this manual is _____

_____

_____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

For software manuals, please indicate which version of the software you are using: _____

_____

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

---- Do Not Tear - Fold Here and Tape ----

digital™

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
OpenVMS Documentation
110 SPIT BROOK ROAD ZKO3–4/U08
NASHUA, NH 03062–2642

---- Do Not Tear - Fold Here ----

Reader's Comments

POLYCENTER Software Installation
Utility Developer's Guide
AA–Q28MA–TK

Your comments and suggestions help us improve the quality of our publications.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (product works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

_____

What I like best about this manual is _____

_____

_____

What I like least about this manual is _____

_____

_____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

For software manuals, please indicate which version of the software you are using: _____

_____

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

Do Not Tear - Fold Here and Tape

digital™

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
OpenVMS Documentation
110 SPIT BROOK ROAD ZKO3–4/U08
NASHUA, NH 03062–2642

Do Not Tear - Fold Here

digital

Printed in Europe